JAM 7

Configuration Guide

August 1995

This software manual is documentation for JAM[®] 7. It is as accurate as possible at this time; however, both this manual and JAM itself are subject to revision.

JAM is a registered trademark of JYACC, Inc.

PostScript is a trademark of Adobe Systems Incorporated.

VMS, VT100, and VT220 are trademarks of the Digital Equipment Corporation.

DynaText is a trademark of Electronic Book Technologies.

HP is a trademark of Hewlett-Packard Company.

INFORMIX is a registered trademark of Informix Software, Inc.

IBM and Presentation Manager are registered trademarks of International Business Machines Corporation.

SYBASE is a registered trademark of Sybase, Inc.

Windows and ODBC are trademarks and Microsoft and MS-DOS are registered trademarks of Microsoft Corporation.

OSF/Motif is a trademark of the Open Software Foundation.

Sun is a trademark of Sun Microsystems, Inc.

UNIX is a registered trademark in the United States and other countries.

Other product names mentioned in this manual may be trademarks or registered trademarks of their respective owners, and they are used for identification purposes only.

Send suggestions and comments regarding this document to: Technical Publications Manager JYACC, Inc. 116 John Street New York, NY 10038 (212) 267–7722

© 1995 JYACC, Inc. All rights reserved. Printed in USA.

Table of Contents

About this Gu	uide	vii
	Organization of this Guide	vii
	Conventions	viii
	Text Conventions	viii
	Keyboard Conventions	ix
	JAM Documentation	ix
Chapter 1	Introduction	1
	Configuring JAM	1
	Configuration Files	3
	Modifying JAM Configuration Files	3
	Recommendations	4
Chapter 2	Setting Up the JAM Environment	5
-	Setup File	6
	Types of Setup Files	6
	JAM Initialization	8
	Setup File Syntax	9
	Creating or Modifying a Setup File	10
	Converting Setup Files to Binary	10
Chapter 3	Configuration Variables	13
-	Environment Variables	14
	Pointers to Required Files	15
	Pointers to Other Setup Files	16
	Pointers to Application-Specific Files and Information	16

Chapter 4	Setup Variables 19
-	Changing the Default Information/Behavior
	Defining Setup Variables
	Assigning Display Attributes
	Variables for Controlling Behavior
	Cursor Appearance and Movement
	Mouse Cursor Appearance and Behavior (JAM Cursor Only) 2
	Text Selection Appearance
	Function Keys
	Toolbars
	Messages 2
	Shifting and Scrolling
	Character-Mode Label Text Display
	Character-Mode Menus
	Character-Mode Screens 3
	Default Filenames and Extensions 3
	Display Attributes for Grouped Items
	Print Canabilities
	Miscellaneous Setuns 3
	Sample Setup File 3
Chapter 5	Message Files 4 ⁷
	Message Files: Convenient, Flexible and Portable 4
	Advantages of a Message File 4
	Recommendations when Changing and Creating Messages 4
	Message File Syntax 4
	Defining Multiple User Sections 4
	Modifying the Provided Message File 4
	Creating Application Messages 4
	Converting Message Files to Binary 4
	Creating a Header File of User Messages
	Display and Behavior Options in Messages
	Customizing Date and Time Formats
	Date/Time Defaults
	Date/Time Tokens
	Creating Date and Time Defaults
	Literal Dates in Calculations
	Numeric Formats
	Creating a Default Numeric Format
	Decimal Symbols
	-

	Customizing Push Button Labels for Message Boxes Warnings for Character JAM Message Windows Setting Yes/No Values Using Alternate Message Files	70 70 71 71
Chapter 6	Key Translation File	73
-	The Role of the Key Translation File	74
	Viewing Key Sequences	75
	Key Translation File Syntax	77
	Key Mnemonics and Hexadecimal Values	78
	ASCII Character Mnemonics and Hex Values	83
	Creating and Modifying a Key Translation File	83
	Customizing Key Mapping	84
	Using International and Composed Characters	85
	Converting a Key Translation File	86
	Using Alternate Key Translation Files	87
Chanter 7	Video File	80
	The Dole of the Video File	00
	The Pagia Video File	90
	Processing Kaywords Automatic Decomptor Sequencing	90
	Video Eilo Suntov	91
	Inputting Control Characters	92
	Deremeters for Verword Serveroes	93
	Creating and Modifizing a Video Eile	93
	Erre New Terringl	101
	For a New Terminal	.02
	Ennancing a Basic video File	.03
	Ville File Kennede	.04
	Video File Keywords	.05
	Saraan and Lina Eragura	.09 [10
	Curson Desition	.12
		.12 (14
	Display Attributes	.14
	Status Line	13
	Status Line	.24
	Diaphics and International Character Support	.23
	Borders and Line Drawing	.28
		.30
		.32
		.32
	Sample Video Files	.33

Table of Contents

Chapter 8	Configuration Map File	137
	Defining Colors	138
	Color Aliases	138
	Color Schemes	142
	Defining Line and Box Styles	145
	Character Mode	145
	GUI Styles	146
	Defining Display Fonts	147
	Point Sizes	147
	Default Font	147
	Default Font Size	148
	Font Aliases	148
	Converting Configuration Map Files to Binary	150
	Sample Configuration Map File	151
Chapter 9	Setting Windows Defaults	153
-	Initialization Files	153
	Filenames	154
	Syntax of Initialization Files	154
	Colors	155
	Initialization Options	156
	Status Line Appearance	159
	Help Behavior	159
	DDE	160
	Windows Control Panel	160
	Sample JAM.INI File	161
Chapter 10	Setting Motif Defaults	169
•	Resource Files	169
	Resource Filenames	169
	Structure of Resource Files	170
	Location of Resource Files	170
	Colors	171
	Setting Palette Colors	171
	Colors Beyond the JAM Palette	172
	Overriding Colors Set within JAM	172
	Motif Colors	173

Resource Options	174
Behavioral Resources	174
Screen Control Resource	177
Restricted Resources	177
Global Resource and Command Line Options	177
Widget Hierarchy	178
Sample Motif Resource File for JAM	184
Index	189

Table of Contents

About this Guide

The *Configuration Guide* describes the files that are the source of JAM's hardware and software flexibility. Most are found in the JAM configuration (config) directory. The guide also provides instructions on how to modify, create, and compile configuration files to customize JAM for your specific requirements.

You should read this guide if:

- You just installed JAM and want more information about configuring JAM—that is, ascertain the location of the required files and/or directories that are installed and used with JAM.
- You want to set new standards or reset defaults for the behavior of JAM or your application—all setup options are described in this guide.
- You want to adapt or translate the content and style of messages as well as create your own user message files for your application.
- You want to change date/time and currency default formats to comply with individual specifications or international (language-specific) requirements.
- Your JAM application is ready for distribution and you want more information about specifying files and/or directories that JAM will need.

Organization of this Guide

This guide is organized into ten chapters. The first three chapters provide general information about JAM configuration. The next seven chapters discuss the various configuration files:



- Setup file
- Message file
- Key file
- Video file
- Configuration map file
- Windows initialization file
- Motif resource file

Conventions

The following typographical and terminological conventions are used in this guide:

Text Conventions

expression	Monospace (fixed-spaced) text is used to indicate:		
	• Code examples.		
	• Words you're instructed to type exactly as indicated.		
	• Filenames, directories, library functions, and utilities.		
	• Error and status messages.		
KEYWORDS	Uppercase, fixed-space font is used to indicate:		
	• SQL keywords.		
	• Mnemonics or constants as they appear in JAM include files.		
numeric_value	Italicized helvetica is used to indicate placeholders for information you supply.		
[option_list]	Items inside square brackets are optional.		
{x y}	One of the items listed inside curly brackets needs to be selected.		
x	Ellipses indicate that you can specify one or more items, or that an element can be repeated.		

viii

new terms Italicized text is used:

- To indicate defined terms when used for the first time in the guide.
- Occasionally for emphasis.

Keyboard Conventions

XMIT JAM logical keys are indicated with uppercase characters.

Alt+A Physical keys are indicated with initial capitalization, and keys that you press simultaneously are connected with a plus sign.

JAM Documentation

The JAM documentation set includes the following guides and reference material:

Read Me First — Consists of three sections:

- What's New in JAM Briefly describes what's new in JAM 7.
- *Installation Guide* Describes how to install JAM on your specific platform and environment.
- *License Manager Installation* Instructions for installing the License Manager (used on many UNIX and VMS platforms).

Getting Started — Contains useful information for orienting you to JAM. Includes a description of the JAM environment and features, how JAM addresses real-world application development issues, and a guided tutorial for building a mini-JAM database application.

Editors Guide — Instructions about using the JAM authoring environment; learn how to use the graphical tools for creating, editing, and designing your application interface. Includes detailed descriptions of the screen editor, screen wizard, menu bar editor, and styles editor. The *Editors Guide* is also provided online on GUI platforms. It is installed with the installation of the JAM software and can be accessed by selecting help from within the screen editor.

Application Development Guide — Information by topic to guide you in developing your JAM application. This includes components of the JAM development environment such as the repository, hook functions, and menu bars, as well as sections on the SQL executor, SQL generator and the transaction manager.

About this Guide

JAM Documentation

	<i>Language Reference</i> — Describes JPL, JAM's proprietary programming language. Also includes reference sections for JPL commands, built-in functions and JAM library functions. The man pages in the reference sections are arranged alphabetically.			
	<i>Database Guide</i> —Instructions for using JDB, JYACC's prototyping database, and for the commands and variables available in the database interfaces. Includes an Database Drivers section containing instructions unique to each database driver.			
	<i>Configuration Guide</i> — Instructions for configuring JAM on various platforms and to your preferences. Some options that can be set relate to messages, colors, keys and input/output. Also includes information on GUI resource and initialization files.			
	<i>Master Index</i> and <i>Glossary</i> — Provides an index into the entire documentation set and a dictionary of terms used in the documentation set. This is in addition to the indexes in the individual volumes.			
	<i>Upgrade Guide</i> — Online only. Information for upgrading from JAM 5.			
Online Documentation	JAM's documentation set is available online and included with the JAM distribution. The books can be viewed through the DynaText TM browser on GUI platforms. It can be accessed by choosing Help from within JAM or by running DynaText's read-only browser from the command line or by clicking on the DynaText icon. For instructions on using DynaText, request Help while you have a browser window open.			
Collateral	The following information is also provided with your JAM installation:			
Documentation	 Database Driver Notes — JAM 7 has database drivers for most popular relational database engines, as well as JDB, JAM's proprietary database. Information for JDB, Sybase, Oracle, Informix and ODBC are located in the <i>Database Guide</i>; others are included separately. 			
	• Online help — The <i>Editors Guide</i> is provided in online form through the DynaText browser on GUI platforms. It can be accessed by choosing Help from the screen editor. For instructions on using DynaText, request Help while you have a browser window open.			
	• Online README file.			
Additional Help	JYACC provides the following product support services; contact JYACC for more information.			
	• Technical Support			
	• Consulting Services			
	• Educational Services			



Introduction

JAM is designed to be terminal- and system-independent and, therefore, is adaptable and portable across terminals, platforms, databases, and GUI environments. For the most part, once you provide the basic configuration information, JAM and your JAM applications are ready to run.

This introduction includes:

- The meaning of configuring JAM.
- A brief description of configuration files.
- A general method for modifying configuration files—covered in detail in this guide.
- Recommendations for using the contents of the JAM config directory.

Configuring JAM

When you are ready to configure JAM or your JAM application, you must supply the necessary information so that JAM runs correctly on your particular terminal. The installation notes provided with your installation walk you through the actual configuration. In brief, you provide:

JAM requirements

 A value for the SMTERM variable. The value of SMTERM is the name (mnemonic) of the terminal-type you are supporting or emulating if it is different from your system TERM setting. JAM uses the SMTERM or TERM setting to identify

- The name of the binary video file (*vid.bin) that tells JAM how to drive the display.
- The name of the binary key translation file (*keys.bin) that tells JAM how to map the character sequences produced by the keyboard to JAM logical keys.

You must have a key translation file and a video file for each type of terminal you support.

• The location and name of the binary setup file (smvars.bin). The smvars file can tell JAM where to find its binary message file (msgfile.bin) and configuration map file (*cmap.bin).

Table 1 lists the JYACC-distributed ASCII files that are used to configure JAM: the file's purpose, the utility you can use to modify or create the file, the conversion utility you use to convert it to a JAM-usable binary file, and the chapter or chapters containing the details.

Table 1. Files used by JAM

ASCII file	Convert via	Binary file	Create with	Refer to	Purpose
smvars	var2bin	smvars.bin	text editor	Chapter 3 Chapter 4	Contains configuration and setup variables, and provides pointers to JAM-required files
msgfile	msg2bin	msgfile.bin	text editor	Chapter 5	Contains all message text used by JAM as well as date/time and cur- rency format standards
*keys	key2bin	*keys.bin	text editor	Chapter 6	Describes key mapping
*vid	vid2bin	*vid.bin	text editor or term2vid	Chapter 7	Describes terminal-specific capabil- ities and attributes
*map	cmap2bin	*map.bin	text editor	Chapter 8	Application color scheme, screen editor color scheme, color aliases, line style aliases

JAM is compatible with any ANSI terminal, and JYACC provides many key translation and video files (in the config directory) for specific vendor terminals. In the unlikely event that none of the distributed key translation and video files work with your terminal, you can build files from scratch or modify existing ones.

Configuration Files

setting up the environment and controlling JAM and application behavior	JAM applications depend on a number of environment and/or setup variables. The environment variables point to files and directories that provide the necessary information to JAM about the layout of your system and the terminal you are using. The setup files can also serve as a repository for the variables that control many operating parameters in the JAM runtime system and utilities. You can establish new standards or defaults by changing these variables and thereby control certain types of behavior across JAM and your applications. In addition, by using specific library functions with these variables in your code, you can manipulate behavior on an "as needed" basis.
JAM message file	You can adapt and translate the content and style of the messages in the JAM message file to comply with local customs and language, as well as define new default values for date/time and currency formats. These changes can be made to accommodate non-English-speaking JAM developers or to adapt your JAM applications for international distribution.
key translation files	JAM is distributed with key translation files (paired with video files) that are designed to map most vendors' terminal keyboards to JAM's logical keys. If one of these files does not define your keyboard, you can customize a key translation file for your particular needs and terminal keyboard.
video files	JAM is distributed with video files (paired with key translation files) to support most vendors' terminals. If your particular terminal is not among those installed in the config directory, you can modify one of them to support your terminal.
configuration map files	JAM is distributed with configuration map files that define color schemes for screens and for each widget type. They also define color, line style, and box style aliases. You can modify these files to define your own color schemes and to allow your color and line style assignments to port across environments.

Modifying JAM Configuration Files

For the most part, you can easily modify the files provided with JAM. The JAM configuration directory (config) includes ASCII and binary files which you can use to adapt JAM (refer to Table 1) to your needs.

- 1. Access (or create) the ASCII file using a text editor.
- 2. Make the desired changes and save the file.
- 3. Convert the ASCII file to binary format with the appropriate conversion utility, located in the JAM util directory.

Chapter 1 Introduction

4. Make sure that the appropriate configuration variables point to the binary file, since it is the binary file that JAM uses.

Then JAM and your JAM applications are ready to run.

Recommendations

To ensure that any changes you make to configuration files survive later releases of JAM, you should:

- Create a working directory on your system that essentially emulates or mirrors the JAM directory structure.
- Copy the desired ASCII and binary (*.bin) files from the config directory into your working config directory.
- If configuration files require modifications, alter the copies of the files—setup, message, key translation, video, and configuration map files—to suit your particular development or application requirements.
- Maintain the ASCII version of your application user messages in a file separate from the JAM message file (msgfile). You can combine the messages in a single binary file by using the msg2bin utility with the -o option.



Setting Up the JAM Environment

This chapter describes:

- The role of setup files and how they are used to define your development environment as well as your application environment (page 6).
- How to create and modify a setup smvars file (page 10) and convert it to binary format with the var2bin utility (page 10).

Refer to Chapter 3 for detailed descriptions of environment and configuration variables and Chapter 4 for detailed descriptions of setup variables.

To make the most of this chapter, you should be somewhat familiar with the operating system you are using: for example, how to set environment variables. If you are a:

- DOS user, you should be familiar with the command set and the autoexec.bat file.
- UNIX user, you should be familiar with the command setenv (or set and export if you are a Bourne shell user) and shell files like .login, .cshrc or .profile.
- New JAM user, refer to the installation notes distributed with JAM. The notes provide operating system-specific examples for setting environment variables

needed to run JAM. It also lists and describes the contents of all the JAM subdirectories.

Consult your operating system documentation for specific information on these subjects.

If you are a user of a non-UNIX, non-DOS operating system, see your installation notes for system-specific directions.

Setup File

JAM supports a number of variables that let you store and control many operating parameters in the JAM runtime system and utilities. To configure JAM to the structure and layout of your system, JAM is installed with a single setup source—the smvars file in the config directory. This file contains two types of variables: configuration variables and setup variables.

Configuration Variables

Configuration variables, in general, serve as pointers to:

- Files that are required by JAM: key translation, message, and video files for your particular terminal.
- Alternate setup files that contain additional setup variables.
- Application-specific files and information: repository, screen libraries, LDB initialization files, directories to JAM files.

Setup Variables

You can also store application setup variables in the smvars file. These variables control JAM's behavior as well as your application's behavior. JAM is installed with default settings for all setup variables. However, by adding the setup variables to a setup file, you can establish new defaults that control:

- Message display and behavior.
- The text editor that can be invoked from the JPL procedure window.
- Cursor and key behavior during data entry and on screens in general.
- JAM keys that trigger field validation.

Types of Setup Files

There are two files in which you can place configuration and setup variables:

In the binary file named by the environment variable SMVARS.

Note: If your operating system does not support an environment, you can hard-code the location of the binary file.

JAM is installed with an ASCII smvars file and its binary counterpart in the config directory. It contains multiple SMKEY and SMVIDEO entries, which point to terminal-specific key translation and video files.

In the binary file named by the SMSETUP variable, which can be defined in the either the SMVARS file or in the system environment. This file can be terminal-specific, and can, therefore, include settings belonging to an individual or project.



- Figure 1. Via the environment variable SMTERM, JAM can determine which files to use as well as what setup file contains terminal-specific information.
- *order of precedence* All configuration and setup variables (with the exception of SMVARS) can occur in either file. SMVARS itself cannot be put in a setup file. If a variable occurs in both, the setting in the SMSETUP file takes precedence.

In addition, configuration and setup variables can be specified directly in the system environment, which takes precedence over settings defined in either of the setup files.

Note: You can also place certain configuration variables in Windows initialization files. Any setting established in the initialization file override duplicate settings in SMVARS and SMSETUP files. Refer to Chapter 9 of this guide for more information on setting variables in initialization files.

Chapter 2 Setting Up the JAM Environment

JAM Initialization

Application programs initialize JAM by calling sm_initert. This routine is executed before any screens are displayed, and before input is accepted from the keyboard. It must precede most library function calls. Exceptions are calls that install memory-resident message, key translation, and video files, or which set options.

In general, sm_initcrt (initialization of the display and JAM data structures) proceeds as follows:

- Calls an optional user-supplied initialization routine, which can initialize the character string sm_term. If sm_term contains the terminal type, sm_initcrt proceeds to Step 3.
- 2. Determines the terminal type via the environment variable SMTERM.

If the terminal variable is not found in the environment, it reads the setup file defined by the environment variable SMVARS for the SMTERM variable. If SMTERM is not defined, the system TERM variable is used.

If neither terminal variable is found, JAM prompts for the terminal type. If it is not provided, initialization is terminated.

- 3. Processes the files named by the SMVARS and/or SMSETUP environment variables.
- 4. Reads the binary message file as defined by the SMMSGS variable.

If the SMMSGS file is not found, JAM aborts initialization. Error messages encountered prior to loading the JAM message file are hard-coded. Afterward, all error messages are taken from the message file.

5. Seeks and reads the binary video and keyboard files. These files can be defined in SMVARS, in SMSETUP, or in the system environment.

If JAM cannot determine which files to use, JAM prompts for a terminal type, and JAM retries the entire sequence.

- 6. sm_initcrt initializes the operating system's terminal channel. It is set to "no echo" and non-buffered input, if appropriate.
- 7. Initializes the operating system display (using the initialization string found in the video file) and key translation files.

JAM 7.0 Configuration Guide

initialization errors in file I/O are reported using the C library function errors (system-dependent).

Setup File Syntax

The syntax described here applies to entries included in a setup file or defined in the environment. Each line has the following format:

variable = value

The equal sign is required.

variable

One of the configuration or setup variables. Chapter 3 lists configuration variables; Chapter 4 describes setup variables.

value

A string or another keyword. If a line is too long, continue it on the next line by placing a backslash $(\)$ at the end. Lines beginning with a pound sign (#) are comments and are ignored by var2bin.

Certain variables, notably the JAM configuration files—key translation and video—as distributed, have values that depend on the type of terminal you are using. For those variables, the entries in the setup file take the following format:

variable = (term | term2 | ... | termN) value

The named *variable* uses the file called *value* for terminals of type *term1*, *term2*, etc. For example, the following excerpt is from the smvars file:

SMKEY = (ibm) \$SMBASE/config/ibmkeys.bin SMKEY = (hp|hp2392|hpblk) \$SMBASE/config/hpkeys.bin

If SMTERM is set to ibm, SMKEY is set to /usr/jam/config/ibmkeys.bin.

setting a default It is not necessary to use a terminal mnemonic if you are supporting only one terminal type. Any variables that are not terminal-qualified are initialized. You can provide, along with a number of terminal-specific entries of the same type (e.g., SMKEY files), one entry that is not terminal-qualified. This serves as the default and it must be last in the list.

the \$SMBASE directory When you install JAM, the installation program asks for the base directory for your installation, for example: /usr/jam. This path then gets set in your environment as the SMBASE environment variable. You can use \$SMBASE in your setup file as a convenient way to point to files in the JAM installation hierarchy, for example: \$SMBASE/config/sunkeys.bin is a convenient way to refer to /usr/jam/ config/sunkeys.bin. If you ever move your installation, you then only have to change the value of \$SMBASE to point to the new location.

Chapter 2 Setting Up the JAM Environment

Creating or Modifying a Setup File

To configure JAM to the structure and layout of your system, you can create an smvars and/or smsetup files or modify the JYACC-supplied setup file in the config directory

- 1. Access the ASCII file using a text editor.
- 2. Edit the entries using the appropriate syntax.
- 3. Use the JAM var2bin utility to convert the ASCII file to binary format (see below).
- 4. Ensure that the binary filename is defined in the SMVARS variable. Or, if you create a file containing additional configuration or setup variables, be certain to identify it as the SMSETUP variable.

Converting Setup Files to Binary

	Use the var2bin utility to convert ASCII setup files smvars and smsetup to binary format.
Synopsis	var2bin [-pv] [-e ext] setupfile
Arguments and Options	− p Places the binary output file in same directory as the input file.
	-v Lists the name of each input file as it is processed.
	-e Appends the extension <i>ext</i> to the output file instead of the default bin extension.
	setupfile The name of an ASCII setup file; you can specify more than one input file.
Description	The output of var2bin is a binary file having the name of the file you have specified, with a default extension of bin. You designate this output file to be used as a setup file in either the SMVARS or SMSETUP variables, or in the system environment.
10	JAM 7.0 Configuration Guide

To get a brief description of available arguments and command options, type:

var2bin -h

Errors The following list describes possible errors, their cause, and the corrective action to take:

%s is an invalid name.

Cause: The indicated line did not begin with a valid variable name.

Action: Refer to Chapters 3 and 4 for lists of variable names. Correct the ASCII input file, and run var2bin again.

At least one file name is required.

Cause: You have failed to give an input filename. Action: Retype the command, supplying the ASCII setup filename.

Error opening %s.

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

Missing '='.

Cause: An input line did not contain an equal sign after the variable name. Action: Correct the ASCII input file by inserting the equal sign and run var2bin again.

Unable to allocate memory.

Cause: The utility could not allocate enough memory for its needs. Action: None.

%s is an invalid parameter.

- Cause: An option in the input is misspelled or misplaced, or conflicts with an earlier option.
- Action: Check the valid options listed in Chapter 4. Correct the ASCII input file and run var2bin again.

Chapter 2 Setting Up the JAM Environment

Configuration Variables

This chapter describes the four general categories of configuration variables:

- Environment variables
- Variables that point to files required by JAM
- Variables that point to alternate setup files
- Variables that point to application-specific files and information

All configuration and setup variables, with the exception of SMVARS, are typically defined in the SMVARS file. However, you can also set these variables in an SMSETUP file and in Windows *.ini files (Chapter 9 tells which variables can be set in initialization files). Furthermore, these settings can be overridden by defining them directly at the system environment level.

order of precedence JAM sets variables from the following sources, listed in order of precedence:

- 1. Environment
- 2. Windows initialization files
- 3. The file named by the SMSETUP variable
- 4. The file named by the SMVARS variable

13

You can also use library functions to reset some variables at runtime; for example, sm_msgread opens an additional message file, like your application message file.

Environment Variables

There are three configuration variables for which JAM seeks values in the environment and not in a file. If SMTERM and SMUSER are not defined explicitly, JAM tries to derive appropriate values. In Windows, JAM checks the initialization file *.ini for SMUSER.

If the SMVARS file is not set by the environment, JAM looks for it in these locations, in the following search order:

- 1. \$SMBASE/config/smvars.bin
- 2. \$SMBASE/smvars.bin

Required Variables

JAM needs to know two pieces of information to get started: what terminal-type is it running on and where it can find the information it needs to interpret input and output. Your system environment can include two entries to answer these questions; for example:

SMVARS = /usr/appl/config/smvars.bin
SMTERM = vt100

SMVARS

This variable identifies the binary setup file (usually smvars.bin). JAM uses the file named in this variable to direct it to other configuration files and setup information. A typical SMVARS file might contain the following excerpt:

SMKEY	=	(vt vt950) \$SMBASE/config/vtkeys.bin
SMKEY	=	(vt100) \$SMBASE/config/vt100keys.bin
SMVIDEO	=	\$SMBASE/config/vt100vid.bin
SMMSGS	=	/appl/config/msgfile.bin
SMPATH	=	/appl/masks
SMEDITOR	=	vi

The entries with parenthetic items are terminal-specific; JAM uses this mnemonic to find the appropriate files for your terminal (as indicated by the JAM environment variable SMTERM or the system variable TERM).

SMTERM

This variable defines the terminal-type if you want JAM to recognize a terminal type (mnemonic) that is different from that defined in the system TERM variable.

For example, your text editor might work fine with a terminal in VT100 emulation, but you might want JAM to use the features of VT220 emulation; so while TERM is set to VT100, you can set SMTERM = vt220.

variable is not defined it looks for LOGNAME, then USER; if it cannot identify you in

Non-requiredSMUSERVariablesOne environment variable is useful to JAM but not required: SMUSER. When you
are working with multi-user libraries, JAM needs to know your user name. JAM
looks for SMUSER only in the environment. It will not seek this value in the
SMVARS, . ini or XJam file. Motif and DOS users can identify themselves in the
environment, and are likely to have a user name defined that JAM can access
without the need of SMUSER. JAM first looks for user identification from your
configuration management tool, if any; failing that it looks at SMUSER; if this

any of these ways, it prompts you for a user name.

SMBASE

This variable points to the root of your JAM installation. For example:

/usr/jam

Setting this variable in your environment allows your installation to be portable, since you can define other variables that point to files relative to SMBASE. For example, your SMVARS might be defined like this:

\$SMBASE/config/smvars.bin

Pointers to Required Files

SMKEY, SMMSGS, and SMVIDEO are required variables. They name the configuration files used by JAM to describe its operating environment. If JAM cannot find these variables defined or the configuration files named by them, it issues an error message and aborts initialization.

JAM looks for these variables in the file specified by SMVARS. Any values set directly in your environment override duplicate values set in your SMVARS file. Finally, any values set in a Windows *.ini override duplicate values in either the environment or SMVARS.

SMKEY

This variable identifies the binary file containing a key translation table for your terminal. You can include a terminal mnemonic in the entry that matches the terminal type designated in your JAM SMTERM variable or system TERM variable.

Chapter 3 Configuration Variables

SMKEY = (vt100)\$SMBASE/config/vt100keys.bin

Refer to Chapter 6 in this book for details about key translation files, and also the library functions sm_getkey and sm_keyinit in the *Language Reference*.

SMMSGS

This variable identifies the binary file containing messages and other printable strings used by the JAM runtime system and utilities. You can include a terminal mnemonic in the entry that matches the terminal type designated in your JAM SMTERM variable or system TERM variable.

SMMSGS = /usr/appl/config/msgfile.bin

Refer to Chapter 5 in this book for details about message files, and the library functions sm_msg_get and sm_msgread in the *Language Reference*.

SMVIDEO

This variable identifies the binary file containing video control sequences and parameters used by the JAM runtime system. You can include a terminal mnemonic in the entry that matches the terminal type designated in your JAM SMTERM variable or system TERM variable.

SMVIDEO = (vt100 | x100) \$SMBASE/config/vt100vid.bin

Refer Chapter 7 in this book for details about video files, and the library function sm_vinit in the *Language Reference*.

Pointers to Other Setup Files

SMSETUP

Use this variable to identify additional binary setup files containing configuration and/or setup variables. If the SMVARS and SMSETUP files contain identical variables, the settings in the SMSETUP file take precedence. By including this variable, you can conveniently store additional variables that are specific to a particular terminal-type, project, or individual. You can include a terminal mnemonic in the entry that matches the terminal type designated in your JAM SMTERM variable or system TERM variable.

SMSETUP = (xterm) motifsetup.bin

Pointers to Application-Specific Files and Information

SMCOLMAP

This variable points to the binary cmap file. The cmap file defines the default colors, or scheme, for your application, font, color and line style aliases, and the

screen editor color scheme. You can include a terminal mnemonic in the entry that matches the terminal type designated in your JAM SMTERM variable or system TERM variable. In this way you can assign color schemes that will take advantage of GUI-specific colors. Refer to Chapter 8 for more information on creating and modifying configuration map files.

SMDICNAME

Use this variable to identify your development repository. Also refer to the library function sm_dicname in the *Language Reference*.

SMDICNAME = /usr/app/dev.dic

SMEDITOR

Use this variable to indicate the name of the desired text editor to use in JPL procedure windows. The named editor is invoked from the screen editor when the appropriate key is selected. If you do not indicate a value for SMEDITOR, JAM's text editor is available. For example, your entry might look like:

```
SMEDITOR = vi
```

Or using terminal-specific syntax, you can assure that the editor most suited to your environment is run. If you are running in Motif, you will want to spawn a new window for your editor, or it will run in the window which invoked JAM:

```
SMEDITOR = (xterm) "xterm -e vi"
SMEDITOR = (win) c:\windows\write.exe
SMEDITOR = (ibm) edit
```

You can define this variable in a setup file or at the environment level, and you can change it at runtime with the library routine sm_soption.

SMFLIBS

Use this variable to identify the screen libraries that are to remain open while JAM is active. Each open library should have its own entry. Refer to the library functions sm_1_open and sm_r_window in the *Language Reference*.

SMFLIBS = /usr/appl/genlib
SMFLIBS = /usr/me/mylib

SMLDBLIBNAME

Use this variable to supply the name of screen libraries whose contents are to be used as local data block initialization filenames by sm_ldb_init. The files in the library will be loaded and activated at application startup. Filenames are listed on successive lines. Refer to Chapter 9 of the *Application Development Guide* for

Chapter 3 Configuration Variables

information on using LDBs. For information on creating and maintaining libraries, refer to page 561 of the *Application Development Guide*; for information about creating a library member, refer to page 97 of the *Editors Guide*

SMLDBLIBNAME = cust.lib
SMLDBLIBNAME = const.lib

SMLDBNAME

Use this variable to supply a list of screen names to be used as local data block initialization filenames by sm_ldb_init. The listed files will be loaded and activated at application startup after any libraries specified with SMLDBLIBNAME have been loaded. Filenames are listed on successive lines. Refer to Chapter 9 of the *Application Development Guide* for information on using LDBs.

SMLDBNAME = ldb1.jam SMLDBNAME = ldb2.jam SMLDBNAME = ldb3.jam SMLDBNAME = ldb4.jam

SMPATH

Use this variable to list the directories where JAM should search for JAM files at runtime, such as screens and JPL procedures. Place a vertical bar (|) between directory paths; do not include blank spaces. Refer to the library function sm_r_window in the *Language Reference*. You can include a terminal mnemonic in the SMPATH entry that matches the terminal type designated in your JAM SMTERM variable or system TERM variable.

SMPATH = (vt100) /usr/appl/forms | /usr/me/testforms

You can define this variable in a setup file or at the environment level, and you can change it at runtime with the library function sm_soption.

SMVIEWER

Use this variable to specify the viewer for output of reports created with JAM/ReportWriter report browser. If you create PostScript reports, you'll want to set this to a PostScript viewer. If this variable is not set, then SMEDITOR is used.



Setup Variables

This chapter describes all the setup variables that you can define either in your environment, in setup files, and at runtime with specific library functions.

In addition to the variables that point to files and application-specific information, a number of setup variables control the behavior of JAM and your application:

- Cursor appearance and movement (page 22).
- How messages are displayed and acknowledged by the user (page 26).
- Scrolling and shifting in fields and arrays (page 29).
- The appearance of screens (page 33).
- Group handling and attributes (page 34).
- Default file extensions (page 33).
- Miscellaneous processing and display options (page 35).

Setup variables can also establish system-wide behavior for particular function keys (page 25), and a variable that you define as the command to print screens (page 35).

This chapter shows how to accomplish these tasks:

• Establish new default settings for JAM and/or your applications.

19

- Review the options available for these setup variables.
- Change variable settings using a specific library function.

A sample setup file is provided at the end of this chapter which shows the syntax for setting most of the desired variables.

JAM has installed default settings for all setup variables. If these are acceptable, you can skip this chapter for now.

Changing the Default Information/Behavior

To change or include setup variables in a setup file:

- 1. Access the desired ASCII setup file (smvars or an alternate setup file) using any text editor.
- 2. Add or change the desired variables using the syntax for the specific variables described in this chapter or in Chapter 3.
- 3. Convert the ASCII file to binary format with the var2bin utility (refer to page 10).
- 4. If this is a new setup file, define its pathname in the SMVARS or SMSETUP configuration variable.

Defining Setup Variables

in setup files	In general, the format for setting variables (unless otherwise stated) in the setup file is <i>variable</i> = <i>option</i> ; for example:
	IN_BLOCK = OK_BLOCK
at runtime	To change the setting at runtime, use the library routine sm_option unless otherwise stated. The form is sm_option (<i>variable</i> , <i>option</i>); for example:
	<pre>sm_option (IN_BLOCK, OK_BLOCK);</pre>
to determine current setting	To get the current setting for most of setup variables, supply sm_option with an argument of NOCHANGE. For example, this statement returns the current value of IN_BLOCK—either OK_NOBLOCK or OK_BLOCK. :
	retval = sm_option (IN_BLOCK, NOCHANGE);
	<pre>IN_BLOCK—either OK_NOBLOCK or OK_BLOCK. : retval = sm_option (IN_BLOCK, NOCHANGE);</pre>

Note: Many of the variable settings have an OK_ prefix—it identifies that the value refers to keyboard input, or "open_keyboard." It does not refer to the variable as being okay, although it might.

Assigning Display Attributes

Many setup variables take display attributes as parameters.

setting default attributes To define a display attribute, select one color and any other attributes using the defined keywords (refer to Table 2). In a setup file, separate the OR'd attributes with blanks, commas, or semicolons.

MB_BORDATT = REVERSE HILIGHT GREEN
MB_DISPATT = RED, B_WHITE; BLINK

at runtimeTo change default display attributes at runtime, use sm_option unless otherwisestated. Separate the variable from its attributes with a comma and use vertical barsOR attributes together. For example:

sm_option (MB_BORDATT, REVERSE | HILIGHT | GREEN); sm_option (MB_DISPATT, RED | B_WHITE | BLINK);

Table 2.Display attribute keywords

Foreground Color	Background Color	Attribute
	B_HILIGHT	NORMAL_ATTR
BLACK	B_BLACK	BLANK
BLUE	B_BLUE	REVERSE
GREEN	B_GREEN	UNDERLN
CYAN	B_CYAN	BLINK
RED	B_RED	HILIGHT
MAGENTA	B_MAGENTA	DIM
YELLOW	B_YELLOW	
WHITE	B_WHITE	
	B_CONTAINER	

Variables for Controlling Behavior

Setup variables that are defined in a setup file—typically in smvars—affect the entire application. To implement variable settings that are specific to a terminal type, include them in a terminal-specific SMSETUP file.

Chapter 4 Setup Variables

21

If you want to change the application behavior without altering the default, change the setting at runtime with sm_option.

This section describes each of the variables and their respective settings.

Cursor Appearance and Movement

These variables control how the cursor appears and moves. The letter (D) in the description indicates the default setting.

IN_BLOCK	Set Cursor Appearance
OK_NOBLOCK	Cursor occupies one character position in a field. (D)
OK_BLOCK	Current field is changed to reverse video to simulate a large cursor. The cursor occupies the entire field.

IN_HARROW Set Horizontal Arrow Movement

Note: Using the left and right arrow keys usually causes the cursor to move to the field to the left or right of the current field, or as indicated by the setup variable. However, using left and right arrow keys in horizontal scrolling arrays moves the cursor to the next occurrence, and causes the array to scroll to the next available occurrence. Therefore, these variable settings only control cursor movement in fields having a single occurrence. Refer to Chapter 19 in the Editors Guide for information on cursor movement in arrays.

OK_FREE	Free cursor movement.
OK_RESTRICT	The cursor moves left and right in the current field, but it does not leave the field.
OK_COLM	The cursor is positioned to the closest field on the current line.
OK_SWATH	Same as OK_COLM.
OK_NXTLINE	The cursor is positioned to the nearest field in the col- umn closest to the current column. Wrapping is ob- served, if set.
OK_NXTFLD	The cursor is positioned to the field closest to the cur- rent line and column. The calculation uses the diagonal distance, assuming a 5 to 2 aspect ratio.
OK_TAB	Left-arrow backtabs to the end of the previous field, and right-arrow tabs to the first character in the next field. Wrapping is observed if set. The next and previous field properties are not observed. (D)

OK_TABNXT Like OK_TAB, but the next field and previous field properties are observed.

IN_VARROW Set Vertical Arrow Movement

Note: Using the up and down arrow keys usually causes the cursor to move up or down to the next field, or as indicated by the setup variable. However, using up and down arrow keys in scrolling arrays moves the cursor to the next occurrence, and causes the array to scroll to the next available occurrence. Therefore, these variable settings only control cursor movement in fields having a single occurrence. Refer to Chapter 19 in the Editors Guide for information on cursor movement in arrays.

OK_FREE	Free cursor movement.	
OK_RESTRICT	Vertical arrow keys ignored in current field.	
OK_COLM	The cursor is positioned to the nearest field that overlaps the current column. Wrapping is observed, if set.	
OK_SWATH	The cursor is positioned to the closest field that overlaps the swath containing the current field. Wrapping is ob- served if set.	
OK_NXTLINE	The cursor is positioned to the nearest field whose line is closest to the current line. Wrapping is observed, if set. (D)	
OK_NXTFLD	The cursor is positioned to the field nearest the current line and column.	
OK_TAB	Down arrow tabs to the first character in next field; up arrow backtabs to last character in the previous field. The next and previous field properties are not observed.	
OK_TABNXT	Like $\ensuremath{OK_TAB}$, but the next field and previous field properties are observed.	
IN_ENDCHAR	Specify Treatment of Last Character In No Auto Tab Field	
OK_ENDWRITE	Last character in a no auto tab field is repeatedly over- written if the cursor is in overwrite mode.	
OK_ENDBEEP	Terminal beeps when user attempts to overwrite last character in a no auto tab field. (D)	
IN_RESET	Set Options for Field-reset	
<i>Note:</i> IN_RESET <i>is ignored on word-wrapped fields.</i>		

Arrow keys can enter the middle of a field.

Chapter 4 Setup Variables

OK_NORESET

23

OK_RESET	When field is entered, cursor always goes to first char- acter position, based on justification and punctuation properties.
OK_TO_END	When field is entered, cursor always go to the first avail- able blank after (or before in the case of right-justifica- tion) the data.(D)
IN_VALID	Set Conditions for Validation On Field Exit
OK_VALID	Validation is performed whenever field is exited (NL, TAB, BACKTAB, arrows, mouse click, etc.).
OK_NOVALID	Validation is performed only when TAB or NL is pressed. Using arrow keys or a mouse click to leave a field does not validate the field. (D)
IN_WRAP	Set Options for Arrow Wrapping
OK_WRAP	Arrow keys wrap. Vertical arrows wrap from top to bot- tom. Right arrows wrap to the beginning of next line (or first line). Left arrows wrap to end of previous line (or last line). (D)
OK_NOWRAP	Arrow keys do not wrap. Terminal beeps if user tries to move the cursor past the edge of the active screen.

Mouse Cursor Appearance and Behavior (JAM Cursor Only)

These variables control mouse cursor appearance and behavior only when the mouse is being run by JAM. In most cases, the mouse is under control of the native environment and mouse behavior is determined by the environment, not JAM.

CLICK_TIME	Define Maximum Click Interval
time	The maximum amount of time between two mouse clicks that defines a double mouse click. Defaults to 250 ms.
MOUS_CRSR_CHAR	Set Mouse Cursor Character
character	Sets the character used to display mouse cursor. Defaults to a block character.
MOUS_CRSR_ATTR	Set Mouse Cursor Attributes
display attributes	Assign the desired attributes for the occurrence under the cursor. These attributes are added to those already assigned to the occurrence. This defaults to REVERSE.
MOUS_CRSR_MASK	Mask Mouse Cursor Attributes
--------------------	---
display attributes	Mask any attributes that should not be added to the cur- sor attributes. If you are assigning a color as the cursor attribute, add NORMAL_ATTR to MOUS_CRSR_MASK. At- tributes of the occurrence are used if they are not masked out.

Text Selection Appearance

The following variables set attributes for text selected in a single or multiline text widget.

TXT_SELECT_ATTR	Set Selected Text Attributes
display attributes	Assign the desired attributes for the selected text. These attributes are added to those already assigned to the text widget. This defaults to HILIGHT REVERSE.
TXT_SELECT_MASK	Mask Selected Text Attributes
display attributes	Mask any attributes that should not be added to the se- lected text. If you are assigning a color as the selected text attribute, add NORMAL_ATTR to TXT_SE- LECT_MASK. Attributes of the occurrence are used if they are not masked out.

Function Keys

SMINICTRL Associate Control Strings with Fun	nction Keys
--	-------------

Each JAM screen contains a table of control strings associated with functions keys. You can also set default control strings for specified function keys either in SMVARS or in the environment. JAM uses absence of a control string for a given function key.

By including multiple SMINICTRL entries in your SMVARS file (or in the environment), you can define system-wide actions for specific function keys. The syntax for including SMINICTRL variables is as follows:

SMINICTRL = function_key = control_string

For example:

SMINICTRL	=	PF1	=	&system_h	ıelp
SMINICTRL	=	PF2	=	^toggle_m	node
SMINICTRL	=	PF7	=	^jm_keys	SPGD
SMINICTRL	=	PF8	=	^jm_keys	SPGU

Chapter 4 Setup Variables

You can programmatically change application control strings at runtime. For more information, refer to page 29 in the *Language Reference*.

To disable a JYACC-supplied default function key, bind it to a control string function that does nothing or one that calls sm_bel.

Toolbars

These variables control the display of toolbars items and their corresponding tooltips. You can define them in a setup file and at runtime with sm_{option} . The letter (D) in the description indicates the default setting.

Messages

These variables control message display. You can define them in a setup file, in the environment, and at runtime with sm_{option} .

Note: The BLANK attribute keyword is ignored for messages.

MESSAGE_WINDOW	Control When Messages Appear in a Window
WHEN_REQUIRED	Messages appear on the status line unless they are too large to fit. Oversize messages appear in a window. (Messages never appear on the status line in GUI envi- ronments). (D)
ALWAYS	Messages always appear in a window unless they are explicitly sent to the status line (e.g. with sm_d_msg_line and sm_msg).
SMSGPOS	Set Position of Message Line
number	Set the position for the message line by specifying a single number (1 is the top line of the display). This variable is ignored if the terminal has a hardware status line.

SMSGBKATT	Set Background Attributes for Message Line
display attributes	Set message line background attribute. The default is SMSGBKATT = B_BLACK. The keywords for SMSGBKATT take the format B_ <i>color</i> . Refer to Table 2 on page 21 for a list of the keywords.
STEXTATT	Set Attributes for Status Message Text
display attributes	Change the default display attribute for field status text. Refer to Table 2 on page 21 for a list of the keywords. The default is STEXTATT = WHITE.

Note: If you change the attributes but do not specify a color, the default color becomes BLACK. For example, if you use the entry STEXTATT = BLINK, status messages display with the foreground attributes BLINK and BLACK. Then, if you use the default message line background (see SMSGBKATT), status messages are not visible because they have black text on a black background. Always specify a foreground or background color when setting attribute for text. If this is not convenient, you can set the variable SMSGBKATT to a color other than B_BLACK.

EMSGATT	Set Attributes for Error Messages Text
display attributes	Change the display attributes of message text displayed with sm_femsg and sm_ferr_reset, and the tag por- tion of sm_fquiet_err and sm_fqui_msg messages. The content of the tag is specified in the message file entry SM_ERROR (the default is ERROR:). The default is EMSGATT = WHITE BLINK HILIGHT B_HILIGHT. Refer to Table 2 on page 21 for a list of the keywords.
	If you change this variable without specifying a fore- ground color, the default foreground color becomes BLACK. Refer to the note on STEXTATT for more in- formation.
QUIETATT	Set Attributes for Quiet Error Message Text
display attributes	Change the display attributes of message text displayed with sm_fquiet_err messages. The default is QUIE- TATT = WHITE. See EMSGATT for changing the attrib- utes of the tag portion of these messages. Refer to Table 2 on page 21 for a list of the keywords.
	If you change this variable without setting a foreground

Chapter 4 Setup Variables

Acknowledging Messages

ng These variables control how your application responds to user input when a message appears. The letter (D) in the description indicates the default setting.

ER_ACK_KEY	Define Error Acknowledgment Key
key	There are no keywords—the value must be specified explicitly. The key can be given as number (in decimal, hex, or octal) representing an ASCII character, as an ASCII mnemonic (SP, SOH, ETX, etc.), as quoted charac- ter ('.', '_', etc.), or as a logical key defined in smkeys.h. The default is ER_ACK_KEY = ' ', the space key. If you define a value other than the spacebar, refer to ER_SP_WIND below.
ER_KEYUSE	Use or Discard Key in sm_ferr_reset
ER_NO_USE	All error messages must be acknowledged by ER_ACK_KEY, which is discarded. Any other keys struck between the time of the message display and the press- ing of the acknowledgment key are also discarded. By default, if the user does not press ER_ACK_KEY, JAM displays an error window. Refer to ER_SP_WIND. (D)
ER_USE	Any keypress acknowledges an error message. The type-ahead buffer is flushed when the message is dis- played, and the acknowledging keypress is saved for data-entry. Because any keypress clears the error mes- sage, the message Please press the space bar is not used. If you set this as the default, you can still force the user to acknowledge selective messages by putting %Md at the beginning of the message text. Refer to Chap- ter 5 for more information.
ER_SP_WIND	Remind User to Acknowledge Message
ER_YES_SPWIND	If ER_KEYUSE= ER_NO_USE, and the user presses another key when ER_ACK_KEY is expected, a window appears. The default message is Please hit the space bar after reading this message from the message file entries SM_P1 and SM_P2. If you are using this option and a key other than the space bar for message acknowledgement, modify the message file entry SM_SP1. (D)
ER_NO_SPWIND	If ER_KEYUSE= ER_NO_USE, and the user presses another key when ER_ACK_KEY is expected, the termi- nal beeps (by calling sm_bel). A visual bell can be used if the video file file has a BELL entry.

Shifting and Scrolling

These variables can help you establish standards for handling scrolling and shifting arrays and fields, and the zoom window. You can define them in a setup file, in the environment, and at runtime with sm_{option} . If you change these defaults at runtime, call sm_{option} before opening the screen. The letter (D) in the description indicates the default setting.

ZM_DISPLAY	Specify Zoom Window Size Preference
ZM_ONSCREEN	Displays expanded zoom window at the display size of the operating system window (for example, in the xterm or jterm window). Useful for character-mode applications.
ZM_MAXIMUM	Displays expanded zoom window at its maximum physical display. Useful for applications running in GUI environment.
ZM_SC_OPTIONS	Set Zoom Scroll Options
ZM_NOSCROLL	No scroll expansion on arrays.
ZM_SCROLL	Scroll the current array and display as many occurrences as possible.
ZM_PARALLEL	Scroll all parallel or synchronized arrays. Display as many occurrences as possible. (D)
ZM_1STEP	Scroll and shift in one step.
ZM_SH_OPTIONS	Set Zoom Shift Options
ZM_NOSHIFT	No shift expansion. Fields shift, but no horizontal zoom- ing takes place.
ZM_SCREEN	Shifting arrays have as many on-screen elements as the previous form, which is the original form if ZM_SC_OP- TIONS = ZM_NOSCROLL is used. Otherwise, ZM_SCREEN displays as many items as possible. All syn- chronized arrays are shifted together. (D)
IND_OPTIONS	Set Shift/Scroll Indicator Options
IND_NONE	No indicators.
IND_SHIFT	Shift indicators only.
IND_SCROLL	Scroll indicators only.
IND_BOTH	Shift and scroll indicators. (D)
SCR_KEY_OPT	Set Scroll Field Priority
SCR_NEAREST	Nearest scrolling array to current field scrolls when scrolling keys are used (PGUP and PGDN). (D)

Chapter 4 Setup Variables

SCR_CURRENT	Scrolling keys (PGUP and PGDN) have no affect unless current field is a scrolling array.
SB_OPTIONS	Set Scroll Options for Virtual Windows
SB_NONE	No scroll bars or corner arrows.
SB_BARS	Show scroll bars. (D)
SB_CORNERS	Show corner arrows.
IND_PLACEMENT	Set Position of Shift and Scroll Indicators
IND_FULL	Full width of field. (D)
IND_FLDENTRY	Left or right corner, according to the field's justification.
IND_FLDLEFT	Left corner of field.
IND_FLDCENTER	Center of field.
IND_FLDRIGHT	Right corner of field.
ZW_BORDSTYLE	Set Border Style for Zoom Windows
NOBORDER	No border.
style number	A number between 0 and 9 indicates a style. Default is 1.
ZW_BORDATT	Set Border Attributes for Zoom Windows
display attributes	Default is ZW_BORDATT = RED HILIGHT B_CON- TAINER. Refer to Table 2 on page 21 for a list of the keywords.

Character-Mode Label Text Display

These variables control the display attributes of label text in character-mode applications. These variables belong to two categories:

- Display of label mnemonics: AC_KEEPATTRS, AC_SETATTRS, and AC_SWATTRS. These variables only affect widgets that have Label properties—for example, dynamic labels and push buttons. For display of label mnemonics in menu items, refer to page 31.
- Greying of text in inactive widgets—that is, widgets that cannot get focus: FE_KEEPATTRS, FE_SETATTRS, and FE_SWATTRS. These variables only affect widgets that have Label properties—for example, dynamic labels and push buttons—and menu items.

You can change these variables at runtime with sm_option.

AC_KEEPATTRS	Define Attributes Retained for Keyboard Mnemonic
display attributes	Indicate those attributes to be retained (AND'd) for key- board mnemonic characters that are emphasized. The default includes all attributes currently assigned. Refer to Table 2 on page 21 for a list of the keywords.
AC_SETATTRS	Set Attributes for Keyboard Mnemonic Character
display attributes	Set display attributes to emphasize keyboard mnemonic characters. The default is none. Refer to Table 2 on page 21 for a list of the keywords.
AC_SWATTRS	Set Attributes for Switched Keyboard Mnemonic
display attributes	Set display attributes that are switched for keyboard mnemonic characters. The default is HILIGHT WHITE for menu items HILIGHT CYAN. Refer to Table 2 on page 21 for a list of valid keywords.
FE_KEEPATTRS	Define Attributes Retained for Grayed Items
FE_KEEPATTRS display attributes	Define Attributes Retained for Grayed Items Indicate those attributes to be retained (AND'd) for the label text of inactive items. The default includes all at- tributes currently assigned. Refer to Table 2 on page 21 for a list of the keywords.
FE_KEEPATTRS display attributes FE_SETATTRS	Define Attributes Retained for Grayed Items Indicate those attributes to be retained (AND'd) for the label text of inactive items. The default includes all at- tributes currently assigned. Refer to Table 2 on page 21 for a list of the keywords. Set Attributes for Inactive Items
FE_KEEPATTRS display attributes FE_SETATTRS display attributes	 Define Attributes Retained for Grayed Items Indicate those attributes to be retained (AND'd) for the label text of inactive items. The default includes all attributes currently assigned. Refer to Table 2 on page 21 for a list of the keywords. Set Attributes for Inactive Items Set display attributes for label text. of inactive items The default setting is none. Refer to Table 2 on page 21 for a list of the keywords.
FE_KEEPATTRS display attributes FE_SETATTRS display attributes FE_SWATTRS	 Define Attributes Retained for Grayed Items Indicate those attributes to be retained (AND'd) for the label text of inactive items. The default includes all attributes currently assigned. Refer to Table 2 on page 21 for a list of the keywords. Set Attributes for Inactive Items Set display attributes for label text. of inactive items The default setting is none. Refer to Table 2 on page 21 for a list of the keywords. Set Display Attributes for Inactive Items

Character-Mode Menus

These variables control the way menus appear and behave in character-mode applications. You can change these variables at runtime with sm_option.

MB_KEEPATTRS	Define Attributes Retained for Keyboard Mnemonic
display attributes	Indicate those attributes to be retained (AND'd) for key- board mnemonic characters that are emphasized. The default includes all attributes currently assigned. Refer to Table 2 on page 21 for a list of the keywords.

Chapter 4 Setup Variables

MB_SETATTRS	Set Attributes for Keyboard Mnemonic Character
display attributes	Set display attributes to emphasize keyboard mnemonic characters. The default is none. Refer to Table 2 on page 21 for a list of the keywords.
MB_SWATTRS	Set Attributes for Switched Keyboard Mnemonic
display attributes	Set display attributes that are switched for keyboard mnemonic characters. The default is HILIGHT WHITE. Refer to Table 2 on page 21 for a list of the keywords.
MB_BORDSTYLE	Set Border Style of Menu Bars
NOBORDER	No border.
style number	Set number between 0 and 9 to indicate a style. Default is 1.
MB_BORDATT	Set Border Attributes of Menu Bars
display attributes	Set border display attributes for menu bars. The default is MB_BORDATT = B_WHITE BLACK. Refer to Table 2 on page 21 for a list of the keywords.
MB_DISPATT	Set Text Attributes in Menu Bars
display attributes	Set display attributes for text in menu bars. The default is MB_DISPATT = B_WHITE BLACK. Refer to Table 2 on page 21 for a list of the keywords.
MB_FLDATT	Set Attributes for Menu Bar Options
display attributes	Set display attributes for unselected menu bar options. The default is MB_FLDATT = B_WHITE BLACK. Refer to Table 2 on page 21 for a list of the keywords.
MB_HBUTDIST	Set Distance Between Horizontal Menu Options
number	Set the distance between menu items on a horizontal menu. Default is 2 columns.
MB_LINES_PROT	Reserve Space for Menu Bar
number	Set the number of top lines reserved for a menu bar. Default is 1.
MB_SYSTEM	Specify Presence of System Menu on Menu Bar
OK_SYSTEM	System menu item (==) appears on menu bar. (D)
NO_SYSTEM	System menu item does not appear on menu bar.

Character-Mode Screens

The following variable determines the style of emphasis desired to indicate the current, or active, screen. The options are drop shadows, graying, or border highlights. Drop shadows appear to cast a shadow from the active screen over underlying screens. Graying changes the display attributes of all screens except the active one: highlights turn off and colors change to monochrome by default. Chapter 7 describes how to set the graying attributes with the video file entries EMPHASIS_KEEPATT and EMPHASIS_SETATT (refer to page 115). Border highlighting turns on the highlight characteristic for the active screen border. An application running in character-mode can use any one style, or drop shadows and graying can be used together.

You can establish the style by including the variable in a setup file and reset it at runtime through sm_option.

EMPHASIS	Specify Emphasis Style
DROPSHADOW	Draw a shadow at the screen's uppermost right and bot- tom edges. The shadow is two columns wide and one line deep. The right shadow starts one space below the screen's upper edge, while the bottom shadow starts two columns from the screen's left edge. The bottom shadow is indented two spaces from the left edge of the screen. The shadow is formed by graying the underlying text.
GRAYBKGD	Gray background screens. Only the active screen retains its original display attributes.
HIBORDER	Highlight the border of the active screen.
NONE	Disable display emphasis. (D)

Default Filenames and Extensions

The following variables control default file extensions. You can define them in a setup file, in the environment, and at runtime with sm_{option} unless otherwise indicated. The letter (D) in the description indicates the default setting.

FCASE	Set Case Sensitivity for Filename Searches
CASE_INSENS	JAM ignores case when searching for a file. This does not have any effect on filenames passed to the operating system; only on filenames in libraries or in memory.
CASE_SENS	Filename searches are case sensitive. (D)

Chapter 4 Setup Variables

SMFEXTENSION	Specify Screen File Extension
extension	Screen file extension is used by the JAM run-time sys- tem and various utilities. The default is operating sys- tem-dependent; it can be jam or none. If you supply an extension, JAM appends (or prefixes) it to any screen name that does not already contain an extension. Use F_EXTSEP to specify a character which separates a file- name and the extension. Use F_EXTOPT to specify the placement.
	To change the string at runtime, use the library function sm_soption (refer to the <i>Language Reference</i>).
F_EXTREC	Recognize Screen and Utility I/O File Extensions
FE_IGNORE	Ignore extensions.
FE_RECOGNIZE	Recognize extensions.
	The default for F_EXTREC is the value of EXTMULTS for the system as defined in smcommon.h.
F_EXTOPT	Placement of Screen and Utility I/O File Extensions
FE_FRONT	Put the extension before the filename.
FE_BACK	Put the extension after the filename. (D)
F_EXTSEP	Specify Screen and Utility I/O File Extension Separator
character	There are no keywords — the value must be specified explicitly. The default is a period (.). A separator char- acter can be a number (decimal, hex, or octal) which represents an ASCII character, an ASCII mnemonic (SOH, ETX, etc.), or as quoted character ('.', '_', etc.).

Display Attributes for Grouped Items

These variables control the attributes of the cursor and selected items in groups. You can define them in a setup file, in the environment, and at runtime with sm_{option} .

GA_CURATT	Set Group Cursor Attributes
display attributes	Assign the desired attributes for the occurrence under the cursor. These attributes are added to those already assigned to the occurrence. This defaults to BLINK B_HILIGHT.

GA_CURMASK	Mask Group Cursor Attributes
display attributes	Mask any attributes that should not be added to the cur- sor attributes. If you are assigning a color as the cursor attribute, add NORMAL_ATTR to GA_CURMASK. Attributes of the occurrence are used if they are not masked out.
GA_SELATT	Set Selected Group Occurrence Attributes
display attributes	Assign the desired attributes for a selected group occur- rence. These attributes are added to those already as- signed to the occurrence. This defaults to HILIGHT REVERSE.
GA_SELMASK	Mask Selected Group Occurrence Attributes
display attributes	Mask any attributes that should not be added to the at- tributes for the selected group occurrence. If you are assigning a color to GA_SELATT, add NORMAL_ATTR to GA_SELMASK. Attributes of the occurrence are used if they are not masked out.

Print Capabilities

SMLPRINT Set Default Prin	t Command
---------------------------	-----------

The SMLPRINT variable can be used to indicate the operating system command to print the file (screen) generated by the local print key (LP). The entry must contain the string %s at the place where the file name should go.

For example, your entry in might look like

SMLPRINT= print %s or SMLPRINT = lpr %s

This variable can be defined in a setup file, overridden by setting it at the system environment, and changed at runtime with the library function sm_soption (refer to the *Language Reference*).

Miscellaneous Setups

These variables control a variety of customization issues. They can be defined in a setup file or in the environment as well as at runtime with sm_{option} . The letter (D) in the description indicates the default setting.

Chapter 4 Setup Variables

CHAR_VAL_OPT	Keystroke Filter Validation Option
CHAR_BEEP	Cause a beep if user keyboard input does not pass char- acter validation as defined in the Keystroke Filter prop- erty for a data entry field. (D)
CHAR_MSG	Display message if user keyboard input does not pass character validation as defined in the Keystroke Filter property for a data entry field.
CLOSELAST_OPT	Exit Base Form Without Exiting Application
OK_CLOSE_LAST	Allow base form to close without exiting the application. (D)
NO_CLOSELAST	Exit application when user exits base form.
DA_CENTBREAK	Set Default Century for Two-Digit Dates
two-digit number	Use this option to specify the breaking year between the twentieth and twenty-first centuries when JAM formats two-digit year to four-digit year specifications. This option lets you specify that all two-digit year entries less than the number specified should be in the twenty-first century. For example, if you specify 45. then all two-digit year entries between 00 and 44 indicate the years 2000 to 2044, while those between 45 and 99 indicate 1945 to 1999. By default, this variable is set to 50; JAM therefore
	assumes that all two-digit years between 0 and 50 speci- fy the years 2000 to 2050, while all two-digit years 51–99 are in the 20th century (1951–99).
DECIMAL_PLACES	Set Default Decimal Places for Math Display
number	Set the default number of decimal places for JPL math display.
PLACES_VARIABLE	Set the number of decimal places for JPL math display to equal the number of significant digits in the number, to a maximum of 257. (D)

DW_OPTIONS	Set Delayed-write Options
DW_ON	Turn on delayed-write. Output from library functions is not sent immediately to the display, but is used to update the image in memory. When it is necessary to update the display (e.g., when input is accepted from the key- board), output is sent to the display one line at a time, and a check is made for keyboard input between each line. If the user presses a key before the update is com- pleted, the key is processed before the remaining lines are displayed. This option makes JAM more responsive, especially at low baud rates. You can force the display of a delayed-write with the library function sm_flush (refer to the <i>Language Reference</i>). (D)
DW_OFF	Turn off delayed-write. The display is not flushed until input is accepted from the keyboard. JAM does not check for input while writing to the display. This option can be useful when debugging an application. If you use this option, you may need to add the entry BUFSIZ to your video file.
ENTEXT_OPTION	Set Screen Entry Processing Option
LDB_FIRST	LDBs are examined first for the value of a field, then the screen, on screen entry. On screen exit, this order is reversed. (D)
FORM_FIRST	Screen is examined first for the value of a field, then LDBs, on screen entry. On screen exit, this order is reversed.
EXPHIDE_OPTION	Set Screen Expose/Hide Option
OFF_EXPHIDE	Process screen entry or exit functions only when screen is explicitly opened or closed.
ON_EXPHIDE	Process screen functions when screen is explicitly opened or closed, when screen is exposed by closing an overlying window, or when screen is hidden by opening an overlying window. (D)
LISTBOX_SELECTION	Enables extended selection in list boxes
EXTENDED_SELECTION	Enables extended selection. (D)
SIMPLE_SELECTION	Allows only one selection at a time.
STARTSCREEN	Specifies the startup screen name
screen-name	The name of the JAM screen that the application first displays. JAM searches the open form libraries and along SMPATH for the named screen. If no screen is found, JAM reports an error.

Chapter 4 Setup Variables

	STARTSCREEN is not supported in the application's ini- tialization/resource file or in the environment. It must be set in the file specified by SMVARS or SMSETUP. You can also specify the start screen by setting start_screen_name in jmain.c or jxmain.c.
WW_COMPATIBLE	Emulate JAM 6 Behavior in Word Wrap Fields (character-mode only)
WW_COMPATIBLE_OFF	Prevent execution of of field exit and entry functions when cursor traverses occurrences within a word wrap field. Cursor movement within the text of a word wrap field stays inside that field. (D)
WW_COMPATIBLE_ON	Execute field exit and entry functions when cursor tra- verses occurrences within a word-wrapped field. This setting emulates JAM 6 behavior.
WWTAB	Set Tab Space in Word Wrap Fields
number	Set the number of spaces to move when the TAB key is pressed in a word wrap multiline text field. The default is 5. On Motif, this option is ignored.
XMIT_LAST	Set NL or TAB to Act Like XMIT on Last Field of Screen (triggering screen validation)
XMIT_NL	NL (New Line) logical key behaves like XMIT on last field of a screen.
XMIT_TAB	TAB logical key behaves like XMIT on last field of a screen.
XMIT_NL_TAB	Makes both NL and TAB behave like XMIT on last field of a screen.
XMIT_DISABLE	NL and TAB respond as defined and do not emulate the XMIT key when used in the last field of a screen. (D)

Sample Setup File

The following sample file illustrates the syntax for setting most of the variables discussed in this chapter and the configuration variables in Chapter 3. These variables can be designated in the environment or, as recommended, in a setup file, and can be altered at runtime with specific library functions.

```
SMKEY = (vt100 | x100) /usr/jam/config/vt100keys.bin
SMLPRINT = print %s
SMMSGS = /usr/config/msgfile.bin
SMPATH = /usr/app/forms /usr/me/testforms
SMSETUP = hpsetup.bin
SMVIDEO = (vt100 | x100)/usr/jam/config/vt100vid.bin
SMINICTRL= PF2 = ^toggle_mode
SMINICTRL = PF3 = &popwin(3,28)
SMINICTRL = XMIT = ^commit all
IN_BLOCK = OK_NOBLOCK
IN_WRAP = OK_WRAP
IN_RESET = OK_NORESET
IN_ENDCHAR = OK_ENDWRITE
IN_VALID = OK_VALID
IN_VARROW = OK_FREE
IN_HARROW = OK_TAB
ER_ACK_KEY = PF12
ER_KEYUSE = ER_NO_USE
ER_SP_WIND = ER_YES_SPWIND
EMSGATT = RED; RED, REVERSE
QUIETATT = CYAN HILIGHT BLINK
STEXTATT = WHITE REVERSE
QMSGATT = CYAN REVERSE
SMSGBKATT = RED
SMSGPOS = 25
ZM_SC_OPTIONS = ZM_SCROLL
IND_OPTIONS = IND_BOTH
IND_PLACEMENT = IND_FLDRIGHT
SB_OPTIONS = SB_CORNERS
ZW_BORDSTYLE = 8
ZW_BORDATT = MAGENTA
SMFEXTENSION = jam
F_EXTREC = FE_RECOGNIZE
F_EXTOPT = FE_BACK
F\_EXTSEP = '.'
DW_OPTIONS = DW_OFF
ENTEXT_OPTION = LDB_FIRST
FCASE = CASE_SENS
```

Chapter 4 Setup Variables



Message Files

The message file contains more than messages in the traditional sense. Think of a message as a default application "constant"—for example, text labels for message window push buttons, currency formats, date and time formats, and JAM error messages. Some of these message constants may not be appropriate for your entire application audience. JAM allows you to modify them to suit your needs. The JAM configuration library provides a file of message defaults, msgfile, along with its binary form.

You can:

- Translate message text and default values into other languages for international distribution of your application.
- Customize default formats for date/time and currency edits to comply with local standards or individual preferences.
- Add your own application messages to JAM's msgfile or to a separate message file.

Note that only those messages which might be relevant to an end user, that is runtime messages, are in msgfile. JAM messages which only a JAM developer might see are not translatable.

The first part of this chapter describes message file syntax and how to make messages accessible to JAM. It includes the following general topics:

• How to read entries in the message file (page 44).

- How to modify the JAM ASCII message file (page 47).
- How to create a message file that contains your application-specific messages (page 48).
- How to convert an ASCII message file to a binary format accessible to JAM with msg2bin (page 49).
- How to create a header file with msg2hdr so your application messages can be found by your functions, JAM's functions, and JPL (page 53).
- How to embed attributes and key names in messages (page 57).

The second part of the chapter details the more specialized use of the message file, and includes the following topics:

- How to customize date and time formats (page 60) and currency formats (page 66).
- JAM support of local and system decimal designations (page 69).
- Defining defaults for message window button labels such as Yes, No, and Cancel (page 70).
- Defining defaults for yes/no text values (page 71).
- Using alternate message files (page 71).

Message Files: Convenient, Flexible and Portable

During initialization, JAM looks for the configuration variable SMMSGS (which can be defined in the environment or in an SMVARS file). This variable gives the full pathname of the binary message file. This file is ordinarily a binary version of msgfile, where message constants or "tags" are assigned message text:

SM_HITANY = Hit any key to continue.

JAM's message tags are then defined in the include file smerror.h so the messages can be found by the functions in the JAM libraries. The following excerpts from smerror.h show how SM_HITANY gets assigned the hex value 0x8027 (integer value 32795):

#define SM_MSGS 0x8
#define SM_MSG_OF (unsigned short)SM_MSGS * 0x1000
#define SM_HITANY (SM_MSG_OF+27)

Advantages of a Message File

There are obvious advantages to storing message text and default formats in a file:

- Messages need not be hard-coded either for JAM or for JAM applications. The messages, once converted to binary format, can be retrieved with library functions (e.g., sm_msg_get). Different parts of an application can use the same message text, which saves space.
- When you develop an application, you can edit and compile message text without having to recompile the application.
- Messages are in one place, so you can easily access and modify them.
- You can translate all message text, date/time formats, and currency formats to comply with local languages and customs. For example, if you create a French version of the message file, the status line messages which identify physical keys with logical values will appear in French. System dates will use French names for the days of the week and months of the year. Formats for date/time and currency edits will be adapted to French standards. It might be useful to translate error messages as well.

Recommendations when Changing and Creating Messages

store application messages separately from JAM messages	Store your application messages separately from the JAM messages. Not only is this more manageable, but you avoid the risk of corrupting JAM messages, and avoid losing your messages with new release of the JAM message file. After you create your ASCII message file, you can concatenate them with the JAM message file via the msg2bin utility (using the $-o$ option). In this way, you can have a single binary file, and your messages are available at initialization when JAM loads msgfile.bin.
work with a copy of the file	If you must change the JAM message file, you should modify a copy of msgfile rather than alter the original. By doing this, you avoid losing your changes with new releases of the JAM message file.
divide application messages into sections	The user message file can consist of up to eight user classes numbered 0 to 7. The distributed message file utilizes classes 8 through 15, but they are not designated in the file. If you wish to add user messages, consider using class designators. Each classes' messages begin with a two-character prefix that you define (refer to page 45). Use section classes and prefixes to divided your messages into useful categories. Message classes can be loaded and unloaded from memory separately or as a unit. If you do not specify a class for your messages, they default to class 0.

Chapter 5 Message Files

Message File Syntax

Each entry in a message file has the following format:

TAG = message_content

TAG

A single word without embedded blanks that can include letters, digits, and underscores. The equal (=) sign following *TAG* is required. Blanks are allowed both before and after the equal sign.

If *TAG* identifies a system message, it is defined in the include file smerror.h, and it begins with a standard prefix (listed below). These prefixes are reserved and can not be used for other messages.

system prefixes

- SM Messages and strings used by the JAM runtime library.
- FM Messages issued by the Screen Editor.
- JM Additional runtime messages used by JAM.
- JX Additional runtime messages used by the Screen Editor.
- UT Messages issued by the JAM utilities.
- DM Messages issued by the JAM database editor.
- TP Messages issued by JAM's transaction monitor interface.
- CA Messages issued by the JAM/CASE interface.

message_content

Any alphanumeric string on a single line—although the string must contain at least one non-numeric character.

embedding leading and trailing spaces Leading and trailing spaces are skipped. Use identical quotes at the start and end of *message_content* to include leading and trailing space as literal blank space. If *message_content* begins and ends with the same quotation character, JAM strips off the quotes when it displays the message.

If the content is longer than one line and you wish it to appear on a continuous line, use a backslash to end the lines that are continued. For example,

PQ_FATALERR = Application unable to post your \ transaction. Contact your system manager.

JAM 7.0 Configuration Guide

forcing a new line	Use backslash-n (\n) to force a new line. Use backslash-backslash ($\\)$ to place the backslash character in your message.					
indicating a key mnemonic	Use the ampersand to indicate a key mnemonic for push buttons. For example the following lets a user press the letter O on the keyboard instead of choosing the Oui (Yes) acknowledgment push button:					
	SM_MB_YESLABEL = &Oui					
	JAM automatically displays long status-line messages in a window so that the entire message is visible.					
	<i>message_content</i> can also contain percent sequences that specify appearance, positioning, and acknowledgment information. Refer to page 57 for information on defining message attributes.					
	Messages that define date/time formats and numeric/currency formats have their own syntax. Refer to page 60 for date/time syntax and page 66 for numeric/currency syntax.					
missing message entry	If there is no <i>TAG</i> for a message, or <i>message_content</i> , is missing from the message file and a call is made to display the message, JAM displays the message section and number from the #define statement. For example, if the entry for SM_HITANY is deleted from the JAM message file, and user input invokes this particular message, the status line displays <8-27> — the value for SM_HITANY from the include file smerror.h.					
comments	You can include comments in the message file by beginning the comment line with a pound sign (#). The msg2bin utility ignores commented lines when it compiles the file.					
Example	The following example illustrates the types of entries that can be found in the JAM message file:					
	<pre>SM_DAYA6 = Fri SM_DAYA7 = Sat SM_MOREDATA = No more data. SM_YES = y SM_NO = n SM_MONL1 = January SM_MONL2 = February SM_0DEF_DTIME = %m/%d/%2y %h:%0M SM_MB_HELPLABEL= &Help SM_YN_ERR = %MuPlease enter %Ky or %Kn into this field. JM_HITACK = %MdHit acknowledge key to continue</pre>					

Defining Multiple User Sections

A message file can utilize up to eight user classes, numbered 0 to 7. Each section can contain up to 65529 characters. Each classes' messages begin with a

Chapter 5 Message Files

two-character prefix that you define. Use section classes and prefixes to divided your messages into useful categories. Message classes can be loaded and unloaded from memory separately or as a unit. If you do not specify a class for your messages, they default to class 0.

The distributed message file utilizes classes 8 through 15, but they are not designated in the file. If you wish to add user messages to this file, consider using class designators.

When specifying messages in an application message file, precede each group of messages with a class indicator in the following form:

"XY" = n

" XY"

A two-character (alphanumeric) ASCII code—you must type in the quotes. You can use this identifier as the code argument (in library functions, such as sm_msgread) when calling for specific classes of user messages. This code is used in the same way as the JAM-specific prefixes and therefore, must be different from those reserved for JAM (SM, JM, JX, FM, UT, DM, CA, TP).

п

A digit between 0 and 7, inclusively, that designates the class (in library functions such as sm_msgread).

For example, a user-defined message file with multiple classes might contain the following entries:

example

```
"U0" = 0
U0_BADVAL = Bad value
U0_WRONGDATE = Date must be with 30 days of current date
"U1" = 1
U1_WRONGRATE = This is not the applicable rate.
```

When msg2bin compiles the above messages, it numbers them 0x0, 0x1, and 0x1000.

You can also convert the ASCII message file to a C header file with the msg2hdr utility. Refer to page 53 for a sample output file.

To make a class of messages memory-resident, read in each class individually. For example:

sm_msgread ("UO", 0, MSG_FILENAME | MSG_NOREPLACE, "umsg.bin")
sm_msgread ("U1", 1, MSG_FILENAME | MSG_NOREPLACE, "umsg.bin")

When a message file is compiled with msg2bin, the utility uses the tags to distinguish between system and user messages. It numbers user-defined messages

consecutively starting with the class number times 0x1000. So the fourth message in user class four would be numbered 0x4004. If you do not use classes, msg2bin starts numbering at zero. (JAM's system message numbers are taken from smerror.h.) As a developer, you must remember to maintain the order of user messages and the assignment of their identifiers.

Modifying the Provided Message File

You can change the contents in the the distributed message file (msgfile) in order to perform these tasks:

- Translate JAM message text to another language—for non-English-speaking developers or for international distribution.
- Change terminology to suit your particular needs while working with JAM.
- Change JAM defaults to suit you development and/or local standards or customs. Refer to page 60 for information about modifying date/time formats and defaults, and page 66 for information about changing currency formats.

If you want to add messages to the file, see the next section.

You can modify a message file in three steps:

1. Access the ASCII version of the message file and make changes using a text editor.

For example, if you translate the JAM message file, as illustrated:

SM_DAYL1 = Domingo SM_DAYL2 = Lunes SM_DAYL3 = Martes SM_DAYL4 = Miércoles SM_DAYL5 = Jueves SM_DAYL5 = Viernes SM_DAYL6 = Sábado

the long names for days of the week will appear in Spanish in those fields that have date edits that use the system clock.

2. Convert the ASCII message file to binary format with the msg2bin utility (refer to page 49 for more information about msg2bin).

If you are simply changing the text of a message, you do not have to recompile the application program if the *TAG* has already been defined in a compiled message file.

Chapter 5 Message Files

Creating Application Messages

	You can create as many message files as your application requires. All message entries must use the same syntax assignment as described on page 44; your message <i>TAG</i> s cannot include any of those prefixes reserved for JAM (SM, FM, JM, JX, UT, DM, TP, and CA). When you convert the ASCII file to binary format, the utility msg2bin automatically numbers user-defined messages consecutively starting with zero.				
	Note: If you are modifying your application message file and plan on appending it to the JAM message file, or if you are planning extensive application messages, consider defining multiple classes. This will ensure that the message file does not exceed size limitations (65529 characters per class).				
advantages of using TAG identifiers	While a prefix is not required in an application-message <i>TAG</i> , it is often useful. All entries with the same prefix (including JAM-specific) can then be loaded into memory with a call to the library function sm_msgread (refer to the <i>Language Reference</i>). Then to invoke a specific message, you must define its <i>TAG</i> in the application program according to the assignment made by msg2bin (or msg2hdr for user messages).				
	To create a message file, or add to an existing message file:				
	1.	1. Create or access the ASCII message file using a text editor.			
	2.	Add the desired entries using the syntax described earlier on page 44. For example:			
		<pre>#user messages US_NOTAVAIL = All copies of this movie are unavailable. US_ACTL = Enter an actors last name. #administrator messages ADM_INVALIDRC = Invalid rating code.</pre>			
refer to page 53 for directions on creating a header file	3.	Convert the ASCII file to binary format with the msg2bin utility.			
	4.	Let JAM know about your new messages. There are several ways of accomplishing this. Either define the <i>TAG</i> s in your application program or in an include file:			
		<pre>#define US_NOTAVAIL 0x0 #define US_ACTL 0x1 #define ADM_INVALIDRC 0x2</pre>			
		Or run the msg2hdr utility to create a C header file of your user messages			

(refer to page 53).

JAM 7.0 Configuration Guide

Defining your message TAGs in JPL If you are going to be accessing your messages with JPL, you need to define your *TAG*s as global JPL variables in your program or in a file which you include:

```
global US_NOTAVAIL (1) 0
global US_ACTL (1) 1
global ADM_INVALIDRC (1) 2
```

Or run the msg2hdr utility to create a file of JPL global variables of your user messages (refer to page 53).

5. If you need to gain access to your messages outside of JPL, you must recompile your application program. Your application might then issue the following calls:

```
sm_quiet_err (sm_msg_get (US_NOTAVAIL));
sm_err_reset (sm_msg_get (ADM_INVALIDRC));
```

Converting Message Files to Binary

Use the msg2bin utility to convert ASCII message files to a binary format for use by JAM library functions. To convert a message file, use the desired options in either of the following formats:

msg2bin [-pv] [-e ext] message-file...
msg2bin [-pv] [-o file] message-file...

To obtain a brief description of available arguments and command options, type

msg2bin -h

The output of msg2bin is a binary file; the utility uses the *TAGs* to distinguish between system and user messages. It numbers user-defined messages consecutively starting with the class number times 0x1000 (refer to page 45 for how to define user classes). If there are no classes defined, user-defined messages are automatically numbered consecutively starting from zero; the definitions of system messages are taken from smerror.h. Be sure to maintain the order of messages and the assignment of their identifiers. Use these identifiers in the application programs to invoke the desired messages at runtime. Then recompile and link any non-JPL source that includes any files that contain newly defined messages.

 Arguments and
 message-file

 Options
 The name of the ASCII file containing named messages. More than one input file may be specified.

Chapter 5 Message Files

Converting Message Files to Binary

	-p	Places each binary output file in the same directory as the corresponding input file.			
	-v	Lists the name of each input message file as it is processed.			
	-e	Appends the given <i>ext</i> (extension) to the output files, rather than the default bin extension. If the -0 option is used, $-e$ is ignored.			
create a single binary file from multiple ASCII files	-0	All output is placed in a single file, whose <i>file</i> name follows the option letter. Use this option to concatenate your user messages to JAM-messages in a single binary file. This option will overwrite an existing binary message file of the same name.			
Errors	The foll take:	lowing list describes possible errors, their cause, and the corrective action to			
	Line too long: %s				
	Cause:	The message has exceeded 1024 characters.			
	Action:	Split the message into two separate messages or find a good copy editor.			
	File '	%s' not found.			
	Cause:	An input file was missing or unreadable.			
	Action:	Check the spelling, presence, and permissions of the file in question.			
	Missing '=' in line: %s				
	Cause:	The line in the message had no equal sign following the tag.			
	Action:	Correct the input and re-run msg2bin.			
	Insuff	icient memory available.			
	Cause:	The utility could not allocate enough memory for its needs.			
	Action:	None.			
	Error	in %s line %d: text after final quote %s			
	Warnin	g in %s line %d: text after final quote %s			
	Cause:	The message_content starts with a quote character (",',', ') but there are characters after the matching terminal quote character. Perhaps a backslash is missing. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors.			
	Action: Fix the offending line and rerun the utility.				
	Warning: no messages in user section $"%.2s"$				
	Cause:	The input file had a user message class indicator line that did not have any user messages after it. That section will not be included in the output file.			
	Action:	Remove the offending class indicator and rerun the utility.			

message tag exceeds 80 characters:%s

Cause: Message name too long. Action: Shorten the message name and re-run the utility.

At least one file name is required.

Cause: No message file was specified.

Action: Specify the name of the message file.

Error processing file %s

Cause: An error was encountered reading or writing the file.

Action: Confirm that the file is available and that the target directory can be written.

Invalid character(s) in -x option.

Cause: The –x option (characters to prefix to the tag) starts with a digit or contain a character that is not an alphanumeric or an underscore.

Action: Specify a valid prefix and re-run msg2hdr.

Error in %s line %d: bad tag %s

Warning in %s line %d: bad tag %s

- Cause: The tag is missing or does not consist entirely of letters, digits and/or underscores. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors.
- Action: Fix the offending tag and re-run the utility. Refer to smerror.h for a list of tags.

Error in %s line %d: duplicate message tag %s

Warning in %s line %d: duplicate message tag %s

- Cause: An earlier line also contained the same system message tag. The current line is skipped. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors.
- Action: Fix the offending line and rerun the utility.

Error in %s line %d: duplicate user section class %s

Warning in %s line %d: duplicate user section class %s

Cause: The user section or two letter prefix on a class indicator line is already in use. Class zero is implicitly used if a user message is encountered before any class indicator lines. An output file will be created and RET_SUC-CESS will be returned if warnings are encountered but no errors. Action: Fix the offending line and rerun the utility.

Chapter 5 Message Files

Error in %s line %d: invalid user message class indicator line %s Warning in %s line %d: invalid user message class indicator line %s Cause: A user class indicator line (which is used to start a new message User Section) is defective. The programs assume that any tag starting with a double quote is one of these. The tag must be a double quote followed by two character alphanumeric code and a double quote. The class indicator must be a digit between 0 and 7. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors. Action: Fix the offending line and rerun the utility. Error in %s line %d: message tag exceeds 80 characters %s Warning in %s line %d: message tag exceeds 80 characters %s Cause: Message name too long. Action: Shorten the message name and re-run the utility. Warning in %s line %d: message tag exceeds 31 characters %s Cause: The tag is longer than 31 characters but is shorter than 80. Action: None. Error in %s line %d: missing final quote &s Warning in %s line %d: missing final quote &s Cause: The message content starts with a quote character (", ', ') but does not contain a matching terminal quote character. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors. Action: Fix the offending message and re-run the utility. Warning in %s line %d: prefix does not match user section class %s Cause: The first two characters of a user message do not match the two characters specified in the most recent user message class indicator line. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors. Action: Fix the offending tag and re-run the utility. Error in %s line %d: section too long Warning in %s line %d: section too long Cause: There are too many messages in a section. For user message sections the total length of section's message content plus 3, multiplied by the number of messages in the section must not exceed 65503 (0xFFDF). Stated mathematically: $(section_content + 3) * (#_of _messages) < 65503.$ Action: Create another user section at the specified line.

Creating a Header File of User Messages

	You use the msg2hdr utility to convert an ASCII message file that contains your application's messages to a C header file. To convert a message file, use the following format:				
	msg2hdr [-n <i>nmbr</i>] [-x <i>pfix</i>] [-s <i>pfix</i>] [-dfjpv] [-o <i>file</i>] <i>message-file</i>				
	The square brackets indicate optional command flags. Do not type the brackets.				
	To obtain a brief description of the available arguments and command options, type				
	msg2hdr -h				
	The output of msg2hdr is a .h file with #define statements for each user message <i>TAG</i> . The messages are numbered sequentially starting with 0x0 to 0xF. The message portion is copied to the header file as a comment.				
Options and Arguments	nessage-file Starts numbering messages with the specified <i>nmbr</i> . If no number is entered, 0 (zero) is used.				
	-x Prepends the specified prefix <i>pfix</i> to the TAG portion of the message.				
	-s Select only message names beginning with the prefix <i>pfix</i> .				
	-d Decimal base in the output header file. Default is base 16 (hexadecimal).				
	Output file may overwrite an existing file.				
	Create a JPL global variable file				
	-p Places the output file in the same directory as the corresponding input file.				
	-v Generates list of the file under creation.				
	-o Directs output to the specified <i>file</i> .				
Example	A user message file with multiple sections might look like this:				
	<pre>"U0" = 0 U0_BADVAL = Bad value U0_WRONGDATE = Date must be within 30 days of current date "U1" = 1 WRONGRATE = This is not the applicable rate</pre>				
sample header file with multiple sections	The output would then look like this:				

Chapter 5 Message Files

#define U0 BADVAL $0 \ge 0$ /* Bad value * / #define UO_WRONGDATE 0x1 /* Date must be within 30 \setminus days of current date */ 0×1000 /* This is not the \ #define WRONGRATE applicable rate */ If you had used the -j options, the output would appear as: global U0_BADVAL(1) = 0 /* Bad value * / $global U0_WRONGDATE(1) = 1$ /* Date must be within 30 \setminus days of current date */ = 4096 /* This is not the \setminus global WRONGRATE(4) applicable rate */ The following list describes possible errors, their causes, and the corrective action Errors to take: Exactly one message file name is required Cause: More than one input message file was specified. Action: Run msg2hdr separately for each message file. Consider using the -n option on the subsequent messages to number the messages consecutively. File '%s' already exists; use '-f' to overwrite. Cause: An output file of the same name already exists. Action: Use the $-\circ$ option to specify a different output name or use the -f option to overwrite the existing header file. Missing '=' in line: %s Cause: The line in the message had no equal sign following the tag. Action: Correct the input and re-run msg2hdr. (If you have already converted the message file to binary, you will need to re-run msg2bin) Insufficient memory available. Cause: The utility could not allocate enough memory for its needs. Action: None. Invalid all-numeric message name '%s' Cause: At least one non-numeric character must be in a message name. Action: Rename the offending message and re-run msg2hdr. (If you have already converted the message file to binary, you will need to re-run msg2bin) If no number is entered, 0 will be used. Cause: You did not provide a number with the -n option and it defaulted to zero. Action: Re-run the utility providing a number. Missing message name for '%s' Cause: The message had no characters before the equal sign. Action: Provide a name for the offending message and re-run msg2hdr. (If you have already converted the message file to binary, you will need to re-run msg2bin)

At least one file name is required. Cause: No message file was specified. Action: Specify the name of the message file. Error processing file %s Cause: An error was encountered reading or writing the file. Action: Confirm that the file is available and that the target directory is writeable. Invalid character(s) in -x option. Cause: The -x option (characters to prefix to the tag) starts with a digit or contain a character that is not an alphanumeric or an underscore. Action: Specify a valid prefix and re-run msg2hdr. Warning: no messages in user section \"%.2s\" Cause: The input file had a user message class indicator line that did not have any user messages after it. Action: Remove the offending class indicator. Error in %s line %d: message tag exceeds 80 characters %s Warning in %s line %d: message tag exceeds 80 characters %s Cause: Message name too long. Action: Shorten the message name and re-run the utility. Error in %s line %d: bad tag %s Warning in %s line %d: bad tag %s Cause: The tag is missing or does not consist entirely of letters, digits and/or underscores. It can also indicate the first character of the tag is a digit. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors. Action: Fix the offending tag and rerun the utility. Refer to smerror.h for a list of tags. Error in %s line %d: duplicate message tag %s Warning in %s line %d: duplicate message tag %s Cause: An earlier line also contained the same system message tag. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors. Action: Fix the offending tag and rerun the utility. Error in %s line %d: duplicate user section class %s Warning in %s line %d: duplicate user section class %s Cause: The user section or two letter prefix on a class indicator line is already in use. An output file will be created and RET_SUCCESS will be returned

if warnings are encountered but no errors.

Action: Fix the offending line and rerun the utility.

Chapter 5 Message Files

Error in %s line %d: invalid user message class indicator line %s

Warning in %s line %d: invalid user message class indicator line %s

- Cause: A user class indicator line (which is used to start a new message User Section) is defective. The programs assume that any tag starting with a double quote is one of these. The tag must be a double quote followed by two character alphanumeric code and a double quote. The class indicator must be a digit between 0 and 7. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors.
- Action: Fix the offending line and rerun the utility.
- Error in %s line %d: message tag exceeds 80 characters %s
- Warning in %s line %d: message tag exceeds 80 characters %s
- Cause: The tag is longer than 80 characters. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors.
- Action: Fix the offending tag and rerun the utility.
- Error in %s line %d: message tag exceeds 31 characters %s
- Warning in %s line %d: message tag exceeds 31 characters %s
- Cause: The tag is longer than 31 characters but is shorter than 80. This causes an error if the -j (JPL) option is selected, otherwise it causes a warning. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors.
- Action: Fix the offending tag and rerun the utility.

Error in %s line %d: prefix does not match user section class

Warning in %s line %d: prefix does not match user section class

Cause: The first two characters of a user message do not match the two characters specified in the most recent user message class indicator line. An output file will be created and RET_SUCCESS will be returned if warnings are encountered but no errors.

Action: Fix the offending tag and rerun the utility.

Display and Behavior Options in Messages

Several percent escapes let you control the content and presentation messages. The character or characters following the percent sign are case-sensitive; type them exactly as shown. This prevents conflicts with percent escapes used by printf and the tokens used by date/time formats. Some percent escapes must appear at the beginning of the message; others are valid only for display in a window or on the status line.

Table 3 summarizes the available percent escape sequences, followed by detailed information about each option.

Percent Escape	Description
%A <i>hhhh</i>	Change display attributes. Valid for status line messages only.
%K	Display key label.
%B	Beep the terminal.
%N	Use a carriage return in the message text. Usage of this option forces message in a pop-up window.
8W	Display message in a pop-up window.
%Md	Force the user to press the acknowledgment key (ER_ACK_KEY) in order to dismiss the error message. This option must precede the message text.
%Mt[<i>time-out</i>]	Force temporary display of message to the status line. JAM automatically dismisses the message after the specified timeout elapses and restores the previous status line display.
%Mu	Force message display to the status line and permit any key- board or mouse input to serve as error acknowledgment. JAM then processes the keyboard or mouse input. This option must precede the message text.

Table 3. Percent escapes for status line messages

%A attr-value— Change display attributes

Valid only for status line messages, you can place %A*attr-value* anywhere in the message text. It changes the display attributes of the text that follows it. *attr-value* is a four-digit hexadecimal value that represents one display attribute or the sum of two or more attributes.

Chapter 5 Message Files

If the string to get the attribute change starts with a hexadecimal digit (0...F), pad *attr–value* with leading zeros to four digits. Refer to Table 4 for valid attribute values.

Note: Monochrome terminals ignore color attributes. However, if you are developing for color terminals, include a color code with the %A. Otherwise, both the foreground and background colors default to black when the %A is not followed by a color code.

Table 4 lists the display attributes and their hexadecimal codes as defined in the include file smattrib.h.

Table 4. Display attributes and hexadecimal codes for status line messages

Foreground Attributes*		Background Attributes	
Attribute Mnemonic	Hex Code	Attribute Mnemonic	Hex Code
REVERSE	0010	B_HILIGHT	8000
UNDERLN	0020		
BLINK	0040		
HILIGHT	0080		
DIM	1000		
Foreground Colors		Background Colors	
BLACK (colors are additive)	0000	B_BLACK	0000
BLUE	0001	B_BLUE	0100
GREEN	0002	B_GREEN	0200
CYAN	0003	B_CYAN	0300
RED	0004	B_RED	0400
MAGENTA	0005	B_MAGENTA	0500
YELLOW	0006	B_YELLOW	0600
WHITE	0007	B_WHITE	0700
NORMAL_ATTR	0007	B_CONTAINER	4000
		(inherit color from container)	

*Attributes are additive. One or more foreground attributes can be added to a background attribute, foreground color and background color.

example

The following message appears in red characters on the default black background with "Warning." in blinking characters.

%B — Beep the Terminal

Place %B in a status line or error message so that the terminal is beeped via sm_bel when the message is displayed. You can configure JAM to send a visible bell, such as flashing the screen, by putting a BELL entry in the video file. Refer to Chapter 7.

%K — Display Key Label

Place %K*logical-key* anywhere in the text of status line and error messages. JAM interprets *logical-key* as a mnemonic defined in smkeys.h. If the key translation file defines a key label for the logical key, the key label replaces the percent sequence in the message text. If there is no key label or no such logical key, %K is stripped off and *logical-key* remains in the message text.



Figure 2. The key label is defined in the key translation file; the message file contains the message text. The result is displayed on the screen.

Note: If %K is used in a status line message, the user can push the corresponding logical key onto the input queue by mouse-clicking on the key label text.

%Md — Force User to Acknowledge Error Message

Place %Md at the beginning of an error message so that the user is forced to press the predefined acknowledgment key ER_ACK_KEY to clear the message. If the user presses any other key, JAM displays an error message or beeps, depending on how setup variable ER_SP_WIND is set. The keypress is not processed as data.

The %Md option corresponds to the default message behavior when setup variable ER_KEYUSE is set to ER_NO_USE. If ER_KEYUSE is set to ER_USE—that is, your application default does not require use of the acknowledgement key—set %Md in a message in order to force the user to press the acknowledgement key to clear a message. For more information about acknowledgement options, refer to page 28.

%Mt[time-out] — Display Transient Error Message

Place %Mt at the beginning a transient status line message. JAM automatically dismisses the message after the specified timeout elapses and restores the previous

Chapter 5 Message Files

status line display. Timeout specification is optional; the default timeout is one second. You can specify another timeout in units of 1/10 second with this syntax:

#(n)

where *n* is a numeric constant that specifies the timeout's length. If *n* is more than one digit, the value must be enclosed with parentheses. For example, this statement displays a message for 2 seconds:

msg emsg "%Mt(20)Changes have been saved to database."

The user can dismiss the message before the timeout by pressing any key or mouse clicking. JAM then processes the keyboard or mouse input.

If the message is too long to fit on the status line, JAM displays the message in a window. In this case, users can dismiss the message only by choosing OK or pressing the acknowledgement key. JAM then discards any keyboard input.

8Mu — Use Any Key to Acknowledge Error Message

Valid only for error messages, you must place %Mu at the beginning of an error message. JAM forces message display to the status line and permits any keyboard or mouse input to serve as error acknowledgment. JAM then processes the keyboard or mouse input. In the following example, entering y or n acts as both message acknowledgment and data entry:

%MuPlease enter %Ky or %Kn into this field.

If the message is too long to fit on the status line, JAM displays the message in a window. In this case, users can dismiss the message only by choosing OK or pressing the acknowledgement key. JAM then discards any keyboard input.

%N — Use a Carriage Return in Message Text

Place one or more %Ns anywhere in a message where you want to insert line returns. Use of this option forces message display to a pop-up window.

W — Display Message in a Pop-up Window

Valid only for error messages, you must place %W at the beginning of a message. JAM forces display of the message in a pop-up window.

Customizing Date and Time Formats

The JAM message file includes entries that establish date and time formats and text. It also includes substitution variables that the screen editor uses to assign those formats to fields.
You can modify date/time entries in the message file if you want to

- Translate the text (days of the week and names of months) to comply with local customs.
- Customize the formats (SM_ date/time entries) to comply with local customs or individual preferences.
- Customize the names of format mnemonics and format types for non-English developers or for individual preferences.

This section describes the tags and their default entries and how you can change the entries to meet the needs of both your development environment and your application.

Date/Time Defaults

When you choose Date/Time for the Data Formatting property of a widget, ten default choices for Format Type are made available to you. The message file contains the definitions for these format types: A name tag defines the name of a format type to appear on the Format Type menu, and a corresponding format tag specifies the format associated with that name. For example FM_OMN_DEF_DT defines the name of the first format type as DEFAULT and the corresponding format tag SM_ODEF_DTIME defines its format as m/%d/%2y %h:%0M.

Table 5 lists the date/time name tags as delivered with JAM and their corresponding format type names. These are the names which appear in the screen editor. The entries in Table 6 define the formats that correspond to the date/time tags and names in Table 5. (The tokens in the formats are defined in Table 7.)

Table 5. Default date/time entries

Date/Time Tag	Format Type Name	Formatting Result
FM_0MN_DEF_DT	DEFAULT	4/1/94 13:13
FM_1MN_DEF_DT	DEFAULT DATE	4/1/94
FM_2MN_DEF_DT	DEFAULT TIME	13:13

The tags FM_3MN_DEF_DT through FM_9MN_DEF_DT are undefined in the provided message file. In the screen editor, the format names default to 'Default'. The corresponding formats are defined in the message file, as shown in the table below.

Chapter 5 Message Files

 Table 6.
 Default date/time formats

Date/Time Format Tag	Tokenized Format
SM_0DEF_DTIME	%m/%d/%2y %h:%0M
SM_1DEF_DTIME	%m/%d/%2y
SM_2DEF_DTIME	%h:%OM
SM_3DEF_DTIME	%m/%d/%2y %h:%0M
SM_4DEF_DTIME	%m/%d/%2y %h:%0M
SM_9DEF_DTIME	%m/%d/%2y %h:%0M

Date/Time Tokens

When specifying a format in the message file or as an argument to the library functions sm_sdtime or sm_udtime, you must use some combination of tokens — not the Screen Editor mnemonics like MONL or DEFAULT DATE. In this way, JAM does not need to parse the message file, and the library functions may be used without knowing the names of substitution variables defined or modified in the message file. When JAM performs date calculations using a format, it replaces tokens with their appropriate values. All other characters in the format (i.e., commas, slashes, colons, etc.) are used literally. If you wish to refer to one of the default format types, there are format tokens ranging from %0f to %9f that correspond to each of the format tags (SM_ date/time entries).

The tokens are listed in Table 7. Most of these substitute a numeric value; message entries are indicated for those that substitute text. For example, %4_Y might substitute 1999, whereas %*m would, depending on the date, substitute one of the values defined by SM_MONL1 through SM_MONL12, perhaps July, perhaps Juillet.

Table 7. Definitions of date and time tokens

Description	Token	Message Entries for Text
Year:		
4 digit	%4y	
2 digit	%2y (Use setup t	file to specify century break)

**Tokens provided so default formats may be used with the library functions sm_sdtime and sm_udtime.

Description	Token	Message Entries for Text
Month:		
numeric (1 or 2 digit)	%m	
numeric (2 digit)	%0m	
abbreviated name (3 char)	%3m	SM_MONA1SM_MONA12
full name	%*m	SM_MONL1SM_MONL12
Day of the Month:		
numeric (1 or 2 digit)	%d	
numeric (2 digit)	80d	
Day of the Week:		
abbreviated name (3 char)	%3d	SM_DAYA1SM_DAYA7
full name	8*d	SM_DAYL1SM_DAYL7
Day of the Year:		
numeric (1-366)	%+d	
Time:		
hour (1 or 2 digit)	%h	
hour (2 digit)	%0h	
minute (1 or 2 digit)	%M	
minute (2 digit)	%0M	
second (1 or 2 digit)	%S	
second (2 digit)	%0s	
AM and PM	%p	SM_AM, SM_PM
Ten Default Formats:		
formats specified in message file entries**	%0f - %9f	SM_0DEF_DTIME to SM_9DEF_DTIME
Other:		
literal percent sign	\$ %	

**Tokens provided so default formats may be used with the library functions sm_sdtime and sm_udtime .

Creating Date and Time Defaults

There are ten date and time entries available for defining formats and the names that specify those formats. The tokens for SM_3DEF_DTIME through SM_9DEF_DTIME are the defined identically to the default for SM_0DEF_DTIME. These additional tags are provided so that you can

Chapter 5 Message Files

	• Create your own date/time formats.			
	0	Name them appropriately.		
	To change or create a default format:			
	1.	Open the ASCII version of the message file with a text editor.		
	2.	Change one of the SM_ date/time entries (SM_0DEF_DTIME through SM_9DEF_DTIME) to define the desired format.		
changing the default format		To change the format associated with the DEFAULT substitution variable, you could change		
		SM_ODEF_DTIME = %m/%d/%2y %h:%MO		
		SM_ODEF_DTIME = %d %*m %4y %h:%M0 %p		
		Date fields that are assigned the DEFAULT substitution variable display date/time in the form: 30 November 1994 3:40 PM		
creating a new format		To define a new format, e.g. month/year, you could change		
		SM_3DEF_DTIME = %m/%d/%2y %h:%M0 to		
		SM_3DEF_DTIME = %3m/%4Y		
		Then create the substitution variable with the corresponding FM_ tag:		
		FM_3MN_DEF_DT = SHORTDATE		
		SHORTDATE would now be on the Format Type menu. Date fields created with the Screen Editor whose format is SHORTDATE would display dates in the form: JUN/1993		
	3.	Convert the ASCII message file to binary format with the msg2bin utility.		
	Not forr and	te: Tokens are provided (refer to Table 7) that correspond to each of the default nats so that these defaults can be used with the library functions sm_sdtime sm_udtime.		
Creating Defaults for Non-English Applications	To app	customize the date and time entries in the JAM message file for non-English lications, you can:		
	0	Translate the text entries which name the days of the week and the months of the year. This text is assigned to the tags SM_DAYA1 SM_DAYA7 (abbreviated names of days), SM_DAYL1 SM_DAYL7 (full names of days), SM_MONA1 SM_MONA12 (abbreviated names of months), SM_MONL1 SM_MONL12 (full names of months).		

JAM 7.0 Configuration Guide

• Change the formats associated with the SM_ date/time tags to comply with local customs.

By translating text and changing formats, date fields whose format is DEFAULT might appear like this:

30 Novembre 1994 15:40

For example, to develop an application for French users, translate the text assigned to SM_DAYA1...SM_DAYA7, SM_DAYL1..SM_DAYL7, SM_MONA1...SM_MONA12, and SM_MONL1...SM_MONL12, like this:

SM DAYA1 = Dim SM_DAYA2 = Lun SM_DAYA3 = Mar . . . SM_DAYL3 = Mardi SM_DAYL4 = Mercredi SM_DAYL5 = Jeudi SM_DAYL6 = Vendredi SM_DAYL7 = Samedi SM_MONA1 = Jan SM_MONA2 = Fév SM_MONA3 = Mar . . . SM_MONL7 = Juillet SM_MONL8 = Août SM_MONL9 = Septembre SM_MONL10 = Octobre SM_MONL11 = Novembre SM_MONL12 = Décembre

This method can be useful if you are distributing the same application to users who speak different languages. A user's SMMSGS variable in the local setup (smvars) file or system environment can specify the name of the appropriate message file and screen libraries. Date/time fields then display the date in a language and format familiar to the user, while all programming code remains independent of the user's language.

Creating Defaults in a Non-English Version of JAM

For example, for French-speaking developers, you could provide these entries:

Chapter 5 Message Files

```
FM_YR4 = ANNÉE4
FM_YR2 = ANNÉE2
FM_MON = MOIS
FM_MON2 = MOIS2
FM_DATE = JOUR
...
```

Given these changes, French-speaking developers using jamdev can create date/time formats using substitution variables in their native language — MOIS2, ANNÉE4, and JOURA, while Spanish-speaking developers might use substitution variables like MES2, AÑO4, and DÍAA.

Literal Dates in Calculations

sm_calc (0,0,'days = @date(1/1/2000) - @date(today)');

For more information about the library function sm_calc, refer to the *Language Reference*.

Numeric Formats

When you choose Numeric for the Data Formatting property of a widget, ten default choices for Format Type are made available to you. The message file contains the definitions for these format types: A name tag defines the name of a format type to appear on the Format Type menu, and a corresponding format tag specifies the format associated with that name. For example SM_OMN_CURR_DEF defines the name of the first format type as Local Currency and the corresponding format tag SM_ODEF_CURR defines its format as ".22,1\$".

You can modify the message file to store ten different default numeric formats. Like the date/time message entries, one entry (SM_0DEF_CURR through SM_9DEF_CURR) defines the format, and a corresponding entry (SM_0MN_CURRDEF through SM_9MN_CURRDEF) defines the name of the format type that appears in the Screen Editor.

Numeric Format	Numeric formats are defined as <i>rmxtpccccc</i> :			
Cymax	r	Radix separator or decimal symbol (usually a period or comma)		
	т	Minimum number of decimal places		
	x	Maximum number of decimal places		
	t	Thousands' separator (i.e., a comma or period; b for a blank)		
	p	Placement of currency symbol $(1 = left, r = right, or m = middle)$		
	ссссс	Currency symbol (up to 5 characters, including blank spaces)		
	-			

To specify leading or trailing blanks in a format, enter blank spaces before or after the currency character(s). The spaces become a part of the currency symbol.

The default format . 22, 1\$ indicates:

- period (or decimal point) as the radix separator
- minimum of two decimal places
- maximum of two decimal places
- comma as the thousands separator
- the currency symbol is placed on the left of the number
- \bigcirc the dollar sign (\$) is the currency symbol

Formats in Provided Message File Table 8 lists the numeric tags as delivered with JAM, their format type name, and the corresponding format tag and the default format. Names are defined only for the first three format types. (Names for format types default to 'default'). The last seven names and formats are for other types you may want to create. Therefore, the last seven format types are defined identically to SM1_DEF_CURR, which is set to two decimal places with a comma as the thousands separator.

Chapter 5 Message Files

Numeric Formats

Numeric Tag	Format Type Name	Corresponding Format Tag	Default Format
SM_0MN_CURRDEF	Local Currency	SM_0DEF_CURR	".22,1\$"
SM_1MN_CURRDEF	2 decimal places with commas	SM_1DEF_CURR	".22,"
SM_2MN_CURRDEF	0 decimal places with commas	SM_2DEF_CURR	".00,"
		SM_3DEF_CURR	".22,"
		SM_4DEF_CURR	".22,"
		SM_9DEF_CURR	".22,"

Table 8. Default message entries for defining numeric formats

For example, SM_OMN_CURRDEF defines the name of the format type as Local Currency, and the corresponding format tag SM_ODEF_CURR defines its format as ".22,1\$". A field with this property would to look like \$5,100.75.

Creating a Default Numeric Format

There are ten numeric format entries available. You can overwrite one of the first three provided formats, or change one of the seven dummy formats. To create a numeric format you need to:

- Define a format tag to equal your own numeric format.
- Define its corresponding numeric tag to equal the name of your newly created format variable.

To change a default numeric format:

- 1. Open the ASCII version of the message file with a text editor.
- 2. Change one of the SM_ numeric entries (SM_0DEF_CURR through SM_9DEF_CURR) to define the desired format

To specify leading or trailing blanks in a format, enter blank spaces before or after the currency character. The spaces become a part of the currency symbol.

To add a format for the French franc, you might make the following change:

```
SM_9DEF_CURR = ',22.r F'
```

JAM 7.0 Configuration Guide

68

example

3. And then add the corresponding SM_ numeric entry.

SM_9MN_CURRDEF = Franc

4. Convert the ASCII message file to binary format with the msg2bin utility.

Using the example illustrated here, currency fields that are assigned the Franc numeric format type property display currency entries in the form: 999.999,999 F.

Creating Defaults in a Non-English Version of JAM In addition to modifying the currency formats to comply with local customs, you can translate the names which appear on the numeric format type property menu for non-English speaking developers. The first three entries are adjacent to each other in the JAM message file, beginning with SM_0MN_CURRDEF and ending with SM_3MN_CURRDEF. SM_4MN_CURRDEF through SM_9MN_CURRDEF are also available variables but not pre-defined in the message file.

For example, for Spanish-speaking developers, you might provide these entries:

SM_0MN_CURRDEF = DINERO SM_1MN_CURRDEF = NUMERO

Given these changes, Spanish-speaking developers using jamdev would see format type choices in their native language.

Decimal Symbols

Via the message file tag SM_DECIMAL you can define a default decimal symbol (or radix separator). When you create a currency field with the Screen Editor, you can specify any decimal symbol (or radix separator). However, the SM_DECIMAL entry enforces a default symbol. If SM_DECIMAL is not specified in the message file, JAM tries to determine the appropriate symbol from the operating system.

JAM accommodates 3 types of decimal symbols. These decimals differ in scope and function.

local

Defined by the message file entry for SM_DECIMAL (default is period (.)). It replaces the system symbol within the scope of a JAM application. If the system and local symbols are different, JAM translates appropriately when interacting with system routines. Use the SM_DECIMAL entry when the system decimal symbol is inappropriate.

system

The character used by the operating system when translating characters to internal values or vise versa (e.g., C routines atof, sprintf, etc.).

Chapter 5 Message Files

Note: Setting the system decimal symbol incorrectly will cause unexpected results when JAM processes numeric values.

field

Defined by a currency edit for a specific field. This symbol is used only for data entry validation and for displaying field values. Use field decimal symbols when you nee to handle multiple decimal conventions within a single application. For more information, refer to page 486 in the *Application Development Guide*.

Customizing Push Button Labels for Message Boxes

The message file tags SM_MB_OKLABEL through SM_MB_HELPLABEL provide the text for message box push buttons.

Note: Microsoft Windows for other languages automatically translate standard push buttons to the appropriate language.

You can change as well as translate push button labels by doing the following:

- 1. Access the ASCII version of the message file with a text editor.
- 2. Change the label text (placing an ampersand before the character you wish to serve as a key mnemonic for the push buttons).
- 3. Convert the ASCII message file to binary format with the msg2bin utility.

For example, if you were converting your application to Spanish, you might include the following in your message file:

SM_MB_OKLABEL	=	&Ok
SM_MB_CANCELLABEL	=	&Cancelar
SM_MB_YESLABEL	=	&Si
SM_MB_NOLABEL	=	&No
SM_MB_RETRYLABEL	=	&Re-intentar
SM_MB_IGNORELABEL	=	&Ignorar
SM_MB_ABORTLABEL	=	A&bortar
SM_MB_HELPLABEL	=	&Ayuda

Warnings for Character JAM Message Windows

Some message windows contain icons to alert users to the nature of the message or warning. JAM supports four icon types: Stop, Question, Warning, and Information.

In character JAM, the message file is searched for the tag that corresponds to the specified icon and its associated text; this text appears as the message box's title bar text. The message file tags SM_MB_STOP through SM_MB_INFORMATION provide this text. To translate these labels, access the ASCII version of the message file with a text editor, change the label text and convert the ASCII message file to binary format with the msg2bin utility. For example, if you were converting your application to Spanish, you might include the following in your message file:

SM_MB_STOP	=	Alto!
SM_MB_QUESTION	=	Pregunta
SM_MB_WARNING	=	Aviso!
SM_MB_INFORMATION	=	InformaciÓn

Setting Yes/No Values

The values associated with the message tags SM_YES and SM_NO can be easily translated or standardized to meet your specific needs. For example, you can translate the value for SM_YES to s (short for sí) for Spanish-speaking users.

Library functions such as sm_is_yes , and properties such as keystroke filter that use the SM_YES and SM_NO entries expect and return the appropriate character as defined in the message file. In the case of a Spanish-speaking user, entering s (for an affirmative response) is recognized, whereas y is ignored.

Using Alternate Message Files

The SMMSGS environment variable specifies the binary file to read into memory at JAM's initialization. If you serve an international market, you can give your users the option of selecting from alternate message files. At runtime the user can set the configuration variable SMMSGS for the binary message file that is appropriate to his/her language.

Alternative files for an application (and for non-English versions of JAM) must be identical in terms of the number and sequencing of all messages (refer to page 48 for information about adding messages).

Chapter 5 Message Files



Key Translation File

To build a JAM application, you must be able to enter ASCII data characters (i.e., A, q, 8, !, 7, ?,], ...) and to indicate certain logical key values to JAM—EXIT, XMIT, PF1, SPF1, etc. Since physical keyboards vary from system to system, JAM uses a key translation file to translate sequences you enter on your physical keyboard into logical keys that JAM understands.

JYACC provides key translation files, in the config directory, that support more than 100 terminal types. One of the distributed key files should work for you. However, you may want to create, modify, or examine the key translation file if you:

- Have a unique keyboard that does not have extended keys corresponding to the JAM logical keys.
- Want to change or define key assignments for specific logical keys.
- Want to assign or change key labels for logical keys.

This chapter describes

- What a key translation file does.
- How to read the entries in the key translation file (page 77).
- How to view the key sequences produced by your keyboard with the showkey utility (page 75).

- How to convert key translation files to binary format with the key2bin (page 86) utility.
- Using alternate key translation files in portable applications (page 87).

The Role of the Key Translation File

During initialization, JAM reads the key translation file specified by SMKEY (in the smvars file) into memory. The value for SMKEY is determined by the terminal type indicated in your SMTERM or system TERM variable. Refer to Chapter 3 for more information about setting the SMKEY variable. Now JAM can recognize how your physical keys map to the logical keys.

JAM logical keys are defined as hexadecimal values in the include file smkeys.h. This file is terminal-independent, while key translation files are terminal-dependent. ibmkeys, vt220keys, wy75keys are a few of the available key translation files.

For the most part, ASCII data key, logical values between 1 and hex FF, require no translation and are not included in the key translation file. However, some keyboards do not have the same number of function keys, and most do not have logical keys named HELP or XMIT (Transmit). Therefore, the key translation file specifies a physical key which transmits a unique code or sequence of codes which, in turn, works as a HELP key. Another key works as the XMIT key, and so on. Logical values between hex 100 and hex 1FF are cursor control and editing keys; values greater than hex 1FF are function keys.

So, 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. JAM converts these characters into logical keys, numbered between 1 and 65535 inclusively, before processing them—hence determining if a data character or control character was received.

If the key input is a control character, JAM searches the key translation file for a sequence beginning with that character. If there is a match, additional characters are read until the entire sequence is found, and returns the logical value. So, JAM is able to interpret XMIT when you press a Do key, or an End key, or whatever physical key is mapped to XMIT in your key translation file.



	If a match is found on the initial character, but not the whole sequence, the entire sequence is discarded. If there is no match at all with the entered control character, it is returned unchanged; this is useful for machines such as IBM PCs that use control codes for displayable characters. Chapter 25 of the <i>Application Development Guide</i> contains a detailed discussion of key translation.
	With a one-to-one mapping there are not enough keys on a commercially available keyboard to represent the entire JAM logical keyboard — and for your application, you may not need to use or map all the logical keys. However, to accommodate its larger logical keyboard, JAM combines two or more physical keys to represent a logical key. For example, on a PC the logical key ZOOM is often mapped to Alt-Z.
learn your key mapping	You can print out the ASCII key translation file that supports your keyboard. Key translation file names begin with a mnemonic for the type of terminal you are using, and end with keys. For example, vt100keys is the key translation file for a VT100 terminal. All key translation files distributed by JYACC adhere to this convention.
	If you use JAM on different terminals with varying operating systems and keyboards, then each terminal must have its own key translation file. If necessary, you can create more than one key translation file for a terminal. In this way you can tailor the key mapping for a particular application.
	Refer to Chapter 25 of the <i>Application Development Guide</i> for a discussion of key processing in JAM applications.

Viewing Key Sequences

As an aid to editing key files, the showkey utility is provided. The showkey utility prints out the key sequence of keys that you enter. To run this utility, type showkey at a system prompt. (This is a JAM executable, and as such, the variables SMVARS and SMBASE must be appropriately defined to run this utility.) The showkey window appears:

Chapter 6 Key Translation File

Showkey Utility r 📁
Most Recent Keys:
ESC [k u CR ESC [f u j jESC [4 5 ~
Please type your keystroke twice:
To end program, type "X" repeatedly.

In order to assure that you are not mistyping your keystrokes, showkey expects you to enter your keystrokes twice. If you do not press the same key sequence twice, it will not display the generated characters. Type x twice in a monospace font to exit the showkey utility.

If you want to bind the keystroke(s) to a JAM logical key, copy the decoded keystroke text to your key translation file. For example if you wanted to assign Shift-F10 on your keyboard to the logical key which starts the debugger, you would:

- 1. Start the showkey utility.
- 2. Enter Shift-F10 twice.

The showkey window displays the character sequence that the keystroke transmits. On one platform, the sequence for Shift-F10 was ESC [$45 \sim$

3. Bind the key label and character sequence to the debugger hot key:

DBUG(Shift-F10) = Esc [4 5 ~

- 4. Run the key2bin utility on your altered key file.
- 5. Type xx to exit the utility.

Key Translation File Syntax

Each entry in a key translation file has the format:

```
logical-key(key-label) = character-sequence
```

Lines beginning with a pound sign # are treated as comments. They are ignored by the key2bin utility.

logical-key

The mnemonic or the hexadecimal value of a JAM logical key. For example, the logical TRANSMIT key is represented by the mnemonic XMIT or by the hex value 0x104. The mnemonics and hex values for all JAM logical keys are defined in smkeys.h. Refer to Table 9 on page 79 for a list of these mnemonics and values.

You can assign the same *logical-key* to two (or more) different *character-se-quences*. For example, in a key translation file for the PC, you can make these entries:

```
HELP(Ctrl-F1) = NUL ^
HELP(Ctrl-Home) = NUL w
```

When these entries are used with a PC, both Ctrl–F1 and Ctrl–Home will execute HELP.

key-label

key-label must be enclosed in parentheses The letter, numeral, character, or character string engraved on a physical keytop. One or more physical key labels may be included inside the parentheses, for example (Alt-F1). *key-label* is optional. It can be accessed at runtime through various library functions and the %K escape in status line messages. (Refer to sm_d_msg_line in the *Language Reference*.) Key labels are often helpful in user messages and prompts. If *key-label* is not specified, the logical key value is used in screen displays.

character-sequence

The sequence of characters, up to six characters not including blanks, produced by a keystroke. JAM translates *character-sequence* to be the logical key on the left of the equal sign.

When a physical key is pressed, it transmits a character code which is unique from the code produced by any other key on the keyboard. Each complete character sequence has only one logical value. Although a sequence may include another as a substring.

Chapter 6 Key Translation File

character sequence substrings and keyboard delay	If you use key sequences that are lead-ins of other sequences, you must assign a timing interval with the video file entry keyword KBD_DELAY. For example, if you have defined one logical value to ESC, another to ESC [F and have set KBD_DELAY to 5, JAM will wait a half of a second after ESC is pressed to resolve the ambiguity. If during that interval [F is not received (or any other identifiable sequence that is prefixed with ESC) JAM passes on the logical value of ESC. For more information on keyboard delay, refer to page 111.
Example	The following example is an excerpt from a key translation file:
	EXIT (F1) = SOH @ CR XMIT (Enter) = SOH O CR TAB = HT BACK = NUL SI BKSP = BS
	<pre># These are the arrow keys RARR = ESC [C LARR = ESC [D UARR = ESC [A DARR = ESC [B</pre>
	<pre># The next entry uses a hex value rather # than a mnemonic for logical-key 0x108 = DEL</pre>

Key Mnemonics and Hexadecimal Values

Table 9 is derived from the include file smkeys.h which defines the JAM logical keyboard. Some entries are required by the Screen Editor and are indicated with double asterisks (**).

Logical Key Mnemonic	Hex Value	Description
CLWIN	0x100	close the current window
SYSMN	0x101	access system menu on active screen/window
CLAPP	0x102	close the application
EXIT	0x103	exit**
XMIT	0x104	transmit**
HELP	0x105	help on field*
FHLP	0x106	help on screen or form
BKSP	0x108	backspace*
TAB	0x109	tab*
NL	0x10a	newline*
BACK	0x10b	backtab*
HOME	0x10c	go to first field on screen*
DELE	0x10e	delete character*
INS	0x10f	insert/overwrite character toggle*
LP	0x110	local print, valid only for character-mode
FERA	0x111	clear field*
CLR	0x112	clear all unprotected*
SPGU	0x113	scroll up a page
SPGD	0x114	scroll down a page
LSHF	0x116	left shift
RSHF	0x117	right shift
LARR	0x118	left arrow*
RARR	0x119	right arrow*
DARR	0x11a	down arrow*
UARR	0x11b	up arrow*
LWRD	0x11c	left to previous word
RWRD	0x11d	right to next word
REFR	0x11e	refresh screen*
EMOH	0x11f	go to last field on screen
INSL	0x120	insert line*
DELL	0x121	delete line*
ZOOM	0x122	zoom on field*
SFTS	0x123	soft key select
MTGL	0x124	toggle menu mode
VWPT	0x125	viewport
MOUS	0x126	indicate mouse event
MNBR	0x127	access menu bar

 Table 9.
 JAM Logical Keyboard — key mnemonics and hexadecimal values

* Recommended entries. **Entries required by jamdev. ***Used in wordwrap fields.

Chapter 6 Key Translation File

Logical Key Mnemonic	Hex Value	Description
CYCL	0x128	cycle through a set of sibling windows
DBUG	0x129	hot key for debugger
WMODE***	0x12a	toggle control code display
WTAB***	0x12b	hard tab
WNL***	0x12c	hard new line
ITSEL	0x12d	item selection screen key
BOLN	0x12e	beginning of line (widget)
EOLN	0x12f	end of line (widget)
MDBL	0x131	mouse double click
OPTDN	0x132	option menu drop down key
BOFD	0x133	beginning of entry field
EOFD	0x134	end of entry field
ADDM	0x135	add mode toggle in list box
EXT	0x136	extend selection to select contiguous list items
EXTD	0x137	extend selection with down arrow in list box
EXTU	0x138	extend seleciton with up arrow in list box
ALSYS	0x83d	access and open system menu

* Recommended entries. **Entries required by jamdev. ***Used in wordwrap fields.

Note: The documentation on error messages and error acknowledgment often refers to an error acknowledgment key whose default is the space bar. Since the space bar is a data entry key, it cannot be used as a logical key. Instead, the key is defined as the setup variable ER_ACK_KEY. You can change the entry in the smvars file, in a setup file, in the system environment, or at runtime with the library function sm_option. Refer to Chapter 4.

Table 10 includes the mnemonics and hexadecimal values for function keys (PF), shifted function keys (SPF), and ALT keys. Table 11 includes the mnemonics and hexadecimal values for application function keys (APP).

PF	Hex	SPF	Hex	ALT	Hex
PF1	0x6101	SPF1*	0x4101	ALTA	0x4103
PF2*	0x6201	SPF2*	0x4201	ALTB	0x4203
PF3*	0x6301	SPF3*	0x4301	ALTC	0x4303
PF4*	0x6401	SPF4*	0x4401	ALTD	0x4403
PF5*	0x6501	SPF5*	0x4501	ALTE	0x4503
PF6*	0x6601	SPF6*	0x4601	ALTF	0x4603
PF7*	0x6701	SPF7	0x4701	ALTG	0x4703
PF8*	0x6801	SPF8	0x4801	ALTH	0x4803
PF9*	0x6901	SPF9	0x4901	ALTI	0x4903
PF10*	0x6a01	SPF10	0x4a01	ALTJ	0x4a03
PF11	0x6b01	SPF11	0x4b01	ALTK	0x4b03
PF12	0x6c01	SPF12	0x4c01	ALTL	0x4c03
PF13	0x6d01	SPF13	0x4d01	ALTM	0x4d03
PF14	0x6e01	SPF14	0x4e01	ALTN	0x4e03
PF15	0x6f01	SPF15	0x4f01	ALTO	0x4f03
PF16	0x7001	SPF16	0x5001	ALTP	0x5003
PF17	0x7101	SPF17	0x5101	ALTQ	0x5103
PF18	0x7201	SPF18	0x5201	ALTR	0x5203
PF19	0x7301	SPF19	0x5301	ALTS	0x5303
PF20	0x7401	SPF20	0x5401	ALTT	0x5403
PF21	0x7501	SPF21	0x5501	ALTU	0x5503
PF22	0x7601	SPF22	0x5601	ALTV	0x5603
PF23	0x7701	SPF23	0x5701	ALTW	0x5703
PF24	0x7801	SPF24	0x5801	ALTX	0x5803
				ALTY	0x5903
				ALTZ	0x5a03

Table 10. Hexadecimal values for function keys, shifted function keys, and ALT keys

*Recommended entries.

Chapter 6 Key Translation File

Key Translation File Syntax

APP	Hex	APP	Hex
APP0	0x6002	APP32	0x4002
APP1	0x6102	APP33	0x4102
APP2	0x6202	APP34	0x4202
APP3	0x6302	APP35	0x4302
APP4	0x6402	APP36	0x4402
APP5	0x6502	APP37	0x4502
APP6	0x6602	APP38	0x4602
APP7	0x6702	APP39	0x4702
APP8	0x6802	APP40	0x4802
APP9	0x6902	APP41	0x4902
APP10	0x6a02	APP42	0x4a02
APP11	0x6b02	APP43	0x4b02
APP12	0x6c02	APP44	0x4c02
APP13	0x6d02	APP45	0x4d02
APP14	0x6de2	APP46	0x4e02
APP15	0x6f02	APP47	0x4f02
APP16	0x7002	APP48	0x5002
APP17	0x7102	APP49	0x5102
APP18	0x7202	APP50	0x5202
APP19	0x7302	APP51	0x5302
APP20	0x7402	APP52	0x5402
APP21	0x7502	APP53	0x5502
APP22	0x7602	APP54	0x5602
APP23	0x7702	APP55	0x5702
APP24	0x7802	APP56	0x5802
APP25	0x7902	APP57	0x5902
APP26	0x7a02	APP58	0x5a02
APP27	0x7b02	APP59	0x5b02
APP28	0x7c02	APP60	0x5c02
APP29	0x7d02	APP61	0x5d02
APP30	0x7e02	APP62	0x5e02
APP31	0x7f02	APP63	0x5f02

Table 11. Hexadecimal values for application function keys

ASCII Character Mnemonics and Hex Values

Table 12 lists two- and three-letter ASCII mnemonics for control and extended control characters. It is derived from the include file smascii.h.

Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex	Mnemonic	Hex
NUL	0x00	DLE	0x10			DCS	0x90
SOH	0x01	DC1	0x11			PU1	0x91
STX	0x02	DC2	0x12			PU2	0x92
ETX	0x03	DC3	0x13			STS	0x93
EOT	0×04	DC4	0x14	IND	0x84	ССН	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		
LF	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			•	

Table 12. ASCII character mnemonics and hexadecimal values

Creating and Modifying a Key Translation File

Creating or modifying a key translation file involves four steps:

- 1. Access the desired ASCII key translation file (or create a new one) using a text editor.
- 2. Edit the ASCII key translation file as required.
- 3. Use key2bin to convert the ASCII file to binary format.
- 4. If this is a new key translation file, include the pathname of the binary file as the value of the SMKEY configuration variable. (The default value for SMKEY is

Chapter 6 Key Translation File

set in the file pointed to by SMVARS. Refer to Chapters 2 and 3 for information on changing configuration variables in setup files.)

Customizing Key Mapping

	You can change the mappings of logical keys for the preferences of users or developers.
example	In the distributed key translation files for the PC, XMIT is mapped to the physical End key, and NL is mapped to the Enter key. The key translation file entries look like this:
	XMIT(End) = NUL O NL(Enter) = CR
	NUL 0 is the character-sequence transmitted by the PC's End key; when JAM receives this sequence, it carries out its XMIT function. Similarly, CR is transmitted by the PC's Enter key and JAM responds appropriately.
	To use the Enter key for XMIT and the End key for NL, you can change the key translation file entries to read:
	XMIT(Enter) = CR NL(End) = NUL O
use sm_keyoption to change default key behavior at runtime	The Enter key then is used for XMIT, and the End key is used where a new line (NL) is needed. Changes made to the key translation file affect the entire application. Or you can use the library function sm_keyoption to change the behavior of logical keys at runtime. Refer to the <i>Application Development Guide</i> for more information.
Accessing Extended Keys on the PC	If you are building applications for PCs with extended keyboards you can use the additional keys and key combinations. By default, the extended keys are not recognized in the distributed PC version of JAM. Before you define these keys, you must add a special flag for each class of key or key combination to the INIT entry in your video file. The classes are listed in Table 13.

Table 13. Classes of keys

Flag	Description
XKEY	Allows access to the F11 and F12 function keys.
WINDOWS	Relates to Microsoft Windows. Specifies that JAM may move the cursor, even if the cursor is invisible.
GRAYKEYS	Distinguishes between gray and white cursor positioning keys. For example, this option allows you to assign one value to the gray up arrow key and another value to the white up arrow key.
MULTISHIFT	Permits the use of sequences using the key combinations: Ctrl-Alt-, Shift-Alt-, and Ctrl-Shift-Alt-

For example, an INIT sequence that includes activation the F11 and F12 keys might look like:

```
INIT = C 0, 7, 2, XKEY
```

You might then define F11 and F12 as supplementary HELP and FHLP keys. (Use the showkey utility to determine the key sequences. Refer to page 75 for a description of the utility). Their definitions in the ASCII key translation file appear as:

HELP(F11) = NUL NEL FHLP(F12) = NUL SSA

Extended keys are documented in the PC video files. The INIT entry is documented in Chapter 7.

Using International and Composed Characters

JAM accepts and interprets 8-bit international characters automatically. Some terminals, however, use character sequences which correspond to international characters. Some terminals also allow the user to compose characters (by programming a key to transmit a specified character sequence). To support either situation:

- 1. Assign the sequence to a hex value in the range $0 \times A0$ to $0 \times FE$ in the key translation file.
- 2. Add a corresponding entry in the GRAPH table of the video file to specify the display (refer to Chapter 7 for information on graphics and international character support). Without a GRAPH entry, the 8-bit character is transmitted as is, and must be interpreted by the terminal.

Chapter 6 Key Translation File

interpreting JAM on different operating systems with different character sets. In addition, you can install a key translation table that controls how keyboard input is mapped to the data in the internal character set. (A video translation table in the video file can map from the internal character set to the screen display; refer to Chapter 7.)

JYACC provides a set of sample translation tables (for example, pc2latin.c translates IBM PC Extended Character set to Latin-1) and code to install them. To install or de-install a key translation table, refer to page 539 in the *Language Reference*.

Converting a Key Translation File

You use the key2bin utility to convert an ASCII key translation file that you have edited or created into binary format for use by applications using the JAM library.

To convert a key translation file, use the following format:

key2bin [-pv] [-e ext] keyfile ...

The square brackets indicate the optional command flags. Do not type the brackets.

To obtain a brief description of available arguments and command options, type

key2bin -h

keyfile

key2bin first tries to open its key translation file with the exact name you enter on the command line; if that fails, key2bin appends keys to the name and tries again. The output file is given the name of the successfully opened key translation file plus the default extension .bin.

To make a key translation file memory-resident use the bin2c utility (refer to page 563 of the *Application Development Guide*) to produce a program source file; then compile that output file and link it with your program. For a complete description, refer to page 523 of the *Application Development Guide*.

Arguments and Options

The name of an ASCII key translation file; more than one key translation file may be included. By convention, the key translation file name is an abbreviation of the terminal's name plus keys. The tag keys helps identify the file as a key translation file; for example, vt100keys is the key translation file for a VT100.

- -p Places the binary files in the same directories as the key translation files.
- -v Lists the name of each key translation file as it is processed.

	-e	Appends the given <i>ext</i> (extension) to the output files, rather than the default bin extension.
Errors	The follo to take:	owing list describes possible errors, their causes, and the corrective action
	Cannot Error Cause:	create '%s' writing '%s' An output file could not be created because of lack of permission or perhaps disk space. Correct the file system problem and retry the operation
	Duplic. Cause: Action:	ate key definition in line:\n '%s' The same character-sequence was assigned to more than one logical key in the specified key translation file. Edit the ASCII file and assign a unique character-sequence to key or keys in question. Then run key2bin again.
	Neithe: Cause: Action:	r '%s' nor '%s' found. A key translation file was missing or unreadable. Check the spelling, presence, and permissions of the file in question.
	Unable Cause: Action:	to allocate memory key2bin could not allocate enough memory for its needs. None.
	No key Cause: Action:	definitions in file '%s' Warning only. The key translation file was empty or contained only comments. None.
	Unknow: Cause: Action:	n mnemonic in line: '%s' The line printed in the message does not begin with a logical key mnemonic. Refer to smkeys.h for a list of mnemonics, and correct the input.
	Extra Cause: Action:	characters in sequence neglected, in line: '%s' The key sequence is longer than the maximum of six characters. Correct the input.
	At leas Cause: Action:	st one file name is required. One or more options was specified but no key translation file names were given. Specify file name(s).

Using Alternate Key Translation Files

Many applications support more than one type of keyboard. With JAM you can provide this support without recompiling the application for each keyboard. Each

Chapter 6 Key Translation File

terminal must have a working key translation file and video file. List the pathnames for the key translation and video files in your application's smvars file. Assuming the SMTERM variable is set correctly, JAM selects the correct key translation file from the smvars file during initialization. Refer to Chapter 2 for more information.

For example, the following excerpt is from the smvars file:

SMKEY = (ibm) /usr/jam/config/ibmkeys.bin SMKEY = (hp|hp2392|hpblk) /usr/jam/config/hpkeys.bin



Video File

JAM is designed to run on many displays with widely differing characteristics. These characteristics greatly affect JAM's display of screens and messages. For example, some displays are 80 columns wide, while others are 132 columns. Similarly, the control sequences used to position the cursor and highlight data on the display are often different from model to model. JAM obtains its display characteristics from a video file.

For your convenience, JYACC provides video files in the config directory that support more than 100 terminal types. One of the distributed video files should work for you.

This chapter describes:

- The role of a video file.
- How to read the entries in a video file as well as the concepts used to interpret them (page 92).
- How to modify a video file for your specific needs (page 101).
- How to convert your ASCII video file to binary format with the vid2bin utility (page 104).
- All the keywords that can be included in a video file (page 105) and commands that are be needed to encode parameters (page 93).

In addition, an annotated video file is included in this chapter (page 133).

The Role of the Video File

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 database, 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, like reverse video and blinking simultaneously. The terminfo database can combine attributes; in practice, unfortunately, the appropriate entries are usually missing.

JAM 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 uses an independent file to describe the terminal specific information. Furthermore, some machines, notably the PC, do not have terminfo capability.

In addition, JYACC developed a set of commands that extend the limited set of commands used by termcap and abbreviates verbose sequences used by terminfo. Both syntaxes are supported. 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 described in this chapter; details and examples are also included.

The Basic Video File

The only required entries in the video file are:

- CUP for positioning the cursor.
- ED for erasing the display.

Although JAM functions with these two entries, they offer limited features; for example, no visual attributes. Speed is also a concern, since the sole purpose of many entries in the video file is to decrease the number of characters transmitted to the terminal.

what you get

In the absence of other entries, JAM assumes a 24-line by 80-column screen. Line 24 is used for status text and error messages, and the remaining 23 are available for screens. The non-display attribute is supported and available. The underline attribute is simulated by underscores placed wherever blanks appear in an underlined field. Clearing a line is done by writing spaces. Borders are available, and consist of printable characters only.

Refer to page 103 for specific details on enhancing a basic video file to include display attributes, such as reverse, underline, etc.

Processing Keywords — Automatic Parameter Sequencing

refer to page 93 for information about specific parameters	A stack is used in processing a keyword in the video file—parameters are kept in a separate list. The stack is initially empty and parameters are generally pushed on the stack as needed. The parameters are ordered and a pointer is used to access them. It initially points to the first one. The stack 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. These types of operations use postfix notation. The operands are pushed, then the operation takes place. So the sequence <code>%p1 %p2 %p3 %+ %*</code> leaves <i>parameter1</i> + (<i>parameter2 * parameter3</i>) on the stack. This same mechanism is used by terminfo.
supporting termcap commands	Termcap commands do not use a stack mechanism. To support them, JAM uses an automatic parameter sequencing scheme where a current index into the parameter list is maintained. When a parameter is needed on the stack
	• The current parameter is pushed and the index is incremented.
	○ 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 translated into terminfo-style.
	Although it is possible to use automatic sequencing and explicit parameter pushes in the same sequence, it is not recommended. Explicit pushes of constants with automatic parameter sequencing, however, is a useful combination. For example:
	REPT= %pl %c ESC F %'?' %p2 %+ %c

Chapter 7 Video File

Video File Syntax

The video file is an ASCII text file that can be edited and created using any text editor. Lines beginning with a pound sign (#) are treated as comments. The commented lines are ignored by the vid2bin utility (used to convert the ASCII file to binary format). All video files distributed by JYACC are documented with comments.

The file consists of many instructions, one per line, and has the following format:

keyword = variable-data

keyword

A single word used to define the instructions. Refer to page 105.

When you add entries to a video file, it is essential that you use the formats described for the specific video instruction. No error checking is done at runtime. The vid2bin utility checks for errors like missing, misspelled, and superfluous keywords, but not for duplicated or conflicting entries.

variable-data

A number, a list of characters, a sequence of characters, or a list of further instructions. The *variable-data* is dependent on the keyword you use. Refer to page 93 for information on keywords that use specific parameters as variable-data.

backslash To continue a logical line on the next physical line, end the first line with a backslash. Do *not* leave a space between the backslash and carriage return. All white space (spaces and tabs) is skipped, except where noted. To enter a backslash as the last character of a line, use two backslashes (without spaces). Thus

text \setminus

Continuation line.

text \\

Ends with a backslash.

text \\\

Backslash and a continuation.

double quotes A double quote " starts a string. 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.

"stty tabs"

Has an embedded space.

stty tabs Does not not have an embedded space.

percent sign (%) The percent sign is a control character (refer to Table 15 on page 95 for a list of percent commands). To enter a literal percent sign, enter it twice (for example, %%).

Inputting Control Characters

There are three ways to input non-printing characters, like control characters, in a video file. You can enter:

- Any character as 0x followed by two hexadecimal digits. For example, use 0x41 for A, or 0x01 for control-A, etc. This method is useful for entering codes in the range 0x80 to 0xff (extended ASCII control characters).
- A caret ^ followed by a letter or symbol to represent control characters in the range 0x01 to 0x1f. Either ^A or ^a can represent SOH (0x01). The symbols are ^[for ESC; ^\ for FS; ^] for GS; ^^ for RS; and ^_ for US.
- Two- and three-character ASCII mnemonics to represent control characters. The documentation provided with terminals often lists such sequences.

Refer to Table 12 on page 83 for a list of mnemonics and hexadecimal values.

Extended ASCII control codes can be transmitted only if the communication line and terminal use eight data bits. If this is not possible, the eight-bit code can be replaced by two seven-bit codes—the first code is ESC (0x1b), the second is 0x40 *less* than the desired eight-bit control character. For example, CSI (0x9b) is replaced by ESC 0x5b, or ESC [. If your video file contains extended ASCII control codes, JAM assumes they can be used; it does not transmit the two-character sequence automatically.

Note: Some computers internally toggle the high bit of a character; ESC on a PRIME is $0 \times 9b$ and CSI is $0 \times 1b$, not vice versa. The numbers given in this guide are always standard ASCII.

Parameters for Keyword Sequences

Certain keywords take values or sequences that cannot be completely specified in advance. For example, the cursor position sequence requires the line and column number before moving. The commands using these sequences are passed extra parameters.

Table 14 lists those keywords that are passed parameters. The number in parentheses is the number of parameters for each keyword.

Chapter 7 Video File

Keyword	Action	Parameters
REPT	repeat sequence (2)	Character 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 (12)	Refer to page 123
ASGR	set area graphics rendition (12)	Refer to page 118

Table 14. Keywords and expected parameters

percent commands Par

Parameters are encoded in sequences by percent commands where the sequence starts with the % symbol. Percent commands

- Cause data to be output, or
- Are used for control purposes.

JAM uses a stack mechanism (similar to that used by terminfo) (refer to page 91 for a description). However, use percent commands with care, since all sequences go through the same processing, even those that do not use runtime arguments. In particular, to enter a percent sign as a literal, you must use %%.

Percent commands are summarized in Table 15. They are organized by function, and their source is indicated (C for termcap; I for terminfo; E for JYACC extended command). Descriptions and examples are provided in subsequent sections.

Percent commands that take a count (represented by a # immediately after a %) do not need to have a count specified. If none is specified, the count is assumed to be 1. For example, %w is equivalent to %1w. (This does not apply to %d.)

Percent Command	Source*	Description
Output Commands (p.	age 97)	
88	C / I	Output a percent sign
8.	С	Output a character
%C	Ι	Output a character
%d	C / I	Output a decimal
%#d	Ι	Output a #-digit decimal, blank filled
%0#d	Ι	Output a #-digit decimal, zero filled, like the termcap %2 which is not supported
8+	С	Add and output a character
8#z	E	Output #(decimal number) binary zeros
% # ₩	E	Wait (sleep) # seconds
%S	E	Issue a system command

Table 15. Percent commands

* C = termcap; I = terminfo; E = JYACC extended command

Chapter 7 Video File

Percent Command	Source*	Description	
Stack Manipulation an	nd Arithme	tic Commands (page 98)	
%p#	Ι	Push parameter #(1 - 12 allowed)	
%′C′	Ι	Push the character constant <i>c</i>	
%{ # }	Ι	Push the integer constant #	
8+	Ι	Add	
8-	Ι	Subtract	
응 *	Ι	Multiple	
%/	Ι	Divide	
%m	Ι	Modulus	
8	Ι	Bitwise OR	
8^^	Ι	Bitwise exclusive OR	
8&	Ι	Bitwise AND	
%=	Ι	Logical EQUAL TO	
8>	Ι	Logical GREATER THAN	
%<	Ι	Logical LESS THAN	
8!	Ι	Logical NOT	
%~	Ι	One's complement	
Parameter Sequencing	and Chan	ging Commands (page 98)	
8 # u	Е	Discard # parameters	
% # ⊳	Е	Back up # parameters	
%i	C / I	Increment the next two parameters	
%r	С	Reverse the next two parameters	
Control Flow Comman	nds (page 99	9)	
%? expr %t then-part %e else-part %;	Ι	Conditionally execute one of two com- mand sequences	
expr %t then-part %e else-part %;	Е	Same effect as previous	
\$#(%)	E	Repeat the sequence # times	
%l(%)	E	Select operations from a list	
Perce	nt Command	Source*	Description
---------------	--	--	---
Term	info Commands	Not Suppor	rted
%P, %	g		Letter variables
\$<#>			Padding (use %#z instead) (page 100)
* C = te	ermcap; I = terminfo	E = JYACC ext	tended command
00	Outputs a liter	al percent sig	jn.
010.	Outputs a character, like printf; this command is supplied for termcap compatibility.		
%C	Outputs a char	acter, like pr	rintf. (Equivalent to %.).
%d	Outputs a decimal, any number of digits, no fill. It has variations that allow for specifying the number of digits, and whether blank-fill or zero-fill is to be used.		
% # d	Outputs a #-digit decimal, blank filled. For example, %3d outputs at mos three decimal digits with blank fill.		
%0 # d	Outputs a #-di three decimal	git decimal, z digits with ze	zero filled. For example, %03d outputs at mozero fill.
%+	Adds and outp following %+ i character, and definitions of sequencing.	outs a charactors added to the the paramete %+ in arithme	er. If the stack is empty, the character e next parameter, the sum is output as a r index is incremented. Also refer to tic commands and automatic parameter
8#z	Outputs the sp usually used for erase screen. T	ecified numb or padding, to The sequence	ber of NUL characters (binary zero). It is to insert a time delay for commands such as \$100z outputs 100 pad bytes to the terminal
% # ₩	Waits (sleeps) UNIX and DC library routine and RESET sec	the specified S platforms, is unavailabl quences. The	# of seconds. (Although supported on all it is not supported on systems where the slee le). It is often used as a time delay for INIT sequence %2w evokes a wait of two seconds.
%S	Issues a system passed to the c string, enclose illustrate two y	n command. command inte the text in si ways of maki	The format is %S (<i>string</i> %) — the string is erpreter for execution. To include spaces in the ngle quotation marks. The following exampleng a system call stty tabs:

Chapter 7 Video File

Output Commands

		%S('stty tabs'%) %S(stty SP tabs%)		
Stack Manipulation	Comma stack are	nds are available to push parameters and constants. Only four levels of e supported, and anything pushed off the end is discarded.		
and Arithmetic Commands	%p#	Pushes parameter #; one to 11 is allowed. For example, the sequence %p2 pushes the second parameter.		
	°, ζ,	Pushes the character constant <i>c</i> . For example, the sequence $\frac{x'x'}{x'}$ pushes the character x.		
	%{ # }	Pushes the integer constant #. For example, the sequence $\{12\}$ pushes the number 12.		
	Various arithmetic and logical operations (e.g., \$+, \$/, \$&, \$>, etc.) are supported. They require one or two operands on the stack. If necessary an automatic push is generated, using the next parameter.			
	%'@' %	% % %c Bitwise OR 3 parameters with @, then output result		
	The automatic parameter sequencing mechanism works well in the above example. Since bitwise OR requires two parameters and there is only one on the stack, a push is performed. No push is required to process %c since an entry already exists on the stack. Thus only three parameters are consumed and the result of the bitwise OR is output.			
	In the following sequence the first parameter is pushed, then a space character $(0x20)$ is pushed. The $+$ command pops and adds these values and puts the answer on the stack. c then pops this value and transmits it to the terminal.			
	%p1 %'SP' %+ %c			
Parameter Sequencing	With aut paramet	tomatic sequencing of parameters, sometimes it is necessary to access the ers in a different order. The following percent commands can be specified:		
Commands	% # b	Backs up the # of parameters by decrementing the parameter index. For example, the following sequences output the same parameter twice:		
		%d %b %d %pl %d %pl %d		
	8#u	Uses up or discards # of parameters by incrementing the parameter index. For example, the following sequences output in reverse order:		
		%u %d %2b %d %p2 %d %p1 %d		

Parameter	Parame	ter changing commands either increment parameters or reverse them
Changing Commands	%i	Increments the next two parameters; however, no output is performed and no parameters are consumed. It is used almost exclusively in termcap cursor positioning sequences. It is passed line and column parameters, with the upper left being $(0,0)$. Many terminals expect the line and column to be relative to $(1,1)$. The following sequence adds one to each parameter and sends it out as decimals: ESC [$i d i d i$
	%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 is 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). For example, this sequence outputs column first, and then line: FS G %r %c %c
Control Flow Commands	Control the num	flow commands conditionally execute command sequences. Some specify aber of times to repeat the sequence, and others select operations from a list.
conditional command	%? expr	*t <i>then-part</i> %e <i>else-part</i> %; This is the general if-then-else clause, which can be abbreviated by omitting the <i>if</i> , thus: <i>expr</i> %t <i>then-part</i> %e <i>else-part</i> %;.
		<i>expr</i> is any sequence including the empty sequence. The %t, which is required, pops a value from the stack and tests it, executing <i>then-part</i> if it is true (non-zero) and <i>else-part</i> otherwise.
		<i>then-part</i> and <i>else-part</i> can be any sequence, including the empty sequence. If <i>else-part</i> is empty, %e can be omitted. They can also contain conditionals, so else-if can be implemented. However, this can produce undecipherable sequences. It is provided for terminfo compatibility. The list command (below) is an alternative.
		If %t finds the stack is empty, it generates an automatic push of the next parameter. %t consumes one parameter; however, the incrementing of the parameter index is delayed until after the entire conditional is executed. A conditional always consumes exactly one parameter, regardless of which branch is taken or of the content of <i>then-part</i> or <i>else-part</i> . If either of those use automatic parameter sequencing, they use a local index. Thus even if they consume 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 this sequence, %t ; %c %;, one parameter is consumed. It outputs ; and a character if the parameter is non-zero, otherwise it skips the parameter.

Chapter 7 Video File

	In th com true	e next example, the constant (binary) 1 is pushed, the parameter is pared with 1, and the boolean value is left on the stack. If the value is nothing is done; otherwise the parameter is output as a decimal.
	\$?	%{1} %p1 %= %t %e %p1 %d %;
	The som	following sequence exhibits a simple "either-or" condition that is etimes used to toggle an attribute on or off. It outputs
	ESC (if th	e parameter is non-zero, and
	ESC) othe	rwise:
	ESC %t (%	e) %;
repeat command	%#(%) Perf para are betv exar thre	orms the same action for several parameters. It is used with automatic meter sequencing, and is almost useless if explicit parameter pushes used. The count is specified after the percent sign. All the commands ween %# (and %) are executed the specified number of times. For nple, the first outputs three decimals, and the second outputs up to e non-zero parameters:
	응3(응3(%d %) %t %d %; %)
list command	%l(%) Ava need form	alable as an alternative to an if-then-else-if construction, but is seldom led. It implements a simple select or case conditional. The general hat is: %1(value1: expr1 %; value2: expr2 %; %)
	The The one is m the o lead elen	values are single character constants representing the various cases. expression is executed if the value matches the top of stack. At most expression is executed (i.e., each case contains a break). If the value issing, the expression is evaluated as a default. For correct operation, lefault case must occur last in the list. The colons do not have a ing percent sign, so no selector can be a colon. The %; after the last eent of the list is not required.
	The and by c para ESC	parameter on the stack (automatically pushed, if necessary) is popped tested against the various cases. The parameter index is incremented ne after the entire list is processed, even if the expressions use meters. The following sequence outputs nothing if the parameter is 0; if it is 1; FS otherwise: %1(0:%; 1:ESC%; :FS %)
Padding	Certain termi terminal docu characters ap sequences, tin	nals (or tty drivers) require extra time to execute instructions. Some imentation specifies the delay required for each command. If random pear on the screen, particularly characters that are part of command ne delays may be required. Delays can be introduced in two ways:
100		

100

- %#w Causes a wait (sleep) for the specified number of seconds. This command is usually only required in terminal initialization or reset sequences. A hard reset of a terminal sometimes requires a sleep of one or two seconds. The following sequence causes a two second sleep after terminal reset: ESC c %2w
- %#z Outputs the specified number of zeros. This command is occasionally needed on the erase display or erase line commands and very rarely in the cursor positioning sequence. The number of zeros to send may range from about five, for very short delays, to several thousand for longer delays. Usually 100 or so is adequate. The following sequence specifies 100 pad zeros after clearing the screen: ESC [J %100z

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 \$<#>.

It is easy to convert to the %#z notation, since at 9600 baud, one character takes one millisecond to output.

Creating and Modifying a Video File

There are two ways to create and modify a video file:

- The easiest way is to modify one of the many video files supplied with JAM. Determine if any of these are for terminals similar to yours. This is very often possible because so many terminals emulate one another.
- Review the documentation that comes with your terminal and alter one of the distributed video files.

Identifying the Right File for Your Terminal JAM is distributed with knowledge of how to talk to around 100 different specific terminals as well as those that emulate other terminals.

1. Identify the binary video file in the smvars file in the config directory that supports your terminal. Do this by looking for the parenthetic mnemonic for your terminal type (as you would for a key translation file). For example

SMVIDEO = (vt100)/usr/jam/config/vt100vid.bin

2. Set the value for the SMTERM variable to the mnemonic; in this example, SMTERM = vt100 or enter the mnemonic when prompted for terminal type when you invoke jamdev.

Chapter 7 Video File

3. Invoke jamdev and check that basic functions work correctly.

If the screen does not clear and/or the characters are positioned incorrectly, you can try a different terminal mnemonic to access a different video file, create a video file from the termcap or terminfo databases (if available), or modify an existing video file. Test video files in this manner until you find one that performs basic requirements (clear screen and proper character display).

Table 16 provides a listing of the more popular and used terminal types. Once you determine the family of terminals (there may be more than one video file to support a single family), you can try each of the files associated with the family of terminals by inputting the appropriate terminal mnemonic in the SMTERM setting.

If the family is	Look for	Set SMTERM to
ANSI/DEC VT	ESC [or CSI in the ASCII video files	vt100
Televideo, Wyse, etc.	ESC * or ESC = in the ASCII video files	wy30;wy50
Hewlett-Packard	Video files supporting HP terminals begin with the letters hp.	hp
Data General	Video files supporting Data General terminals begin with the letters dg	dg214

Table 16. General categories of families of terminals

For a New Terminal

If none of the JYACC-distributed video files come close to your particular terminal type, try to find one that will serve as a starting place for customization:

- 1. Identify the desired video file by setting the SMTERM variable in your environment to the mnemonic (for example, SMTERM = vt100) for the selected video file. (Or leave it unset and type the name in when jamdev prompts you for it.)
- 2. Execute jamdev and retrieve an existing screen or try creating one.
- 3. Check a few basic functions: Does the screen clear? Are characters positioned properly (or even close)?

If things seem to work closely to the way you would expect them to — you probably have a good, modifiable video file.

- 4. If the video file works well, but the keys are not translating properly, modify the key translation file or create a new one that is compatible with the video file (refer to Chapter 6).
- 5. Make a copy of the best working video file you found; name it testvid.
- 6. Edit the ASCII video file using any text editor. Comment out or delete all lines except for the ED (erase display) entry and CUP (cursor position) entry.
- 7. Convert the ASCII file to binary format using vid2bin.
- 8. Edit your smvars file. Add the following entry after the last SMVIDEO entry:

SMVIDEO = testvid.bin

Convert the smvars file to binary (using var2bin).

9. Invoke jamdev and check the basic functions.

Enhancing a Basic Video File

Once you have a basic working video file you can enhance the file to provide greater functionality. Incrementally add and test various features until you have a fully operational video file. The easiest way to add the recommended sequences is to copy them from an existing video file or, if available, from your terminal's documentation.

- 1. Provide cursor movement control characters by adding an entry for CMFLGS (refer to page 112).
- 2. Define display attributes (colors, reverse, underline, etc.) with SGR, ASGR, LATCHATT, AREAATT entries (refer to page 115).
- 3. Add a mechanism for turning the cursor on and off so that menus display properly: CON and COF entries (refer to page 114).
- 4. Consider adding entries to enable the use of graphic characters to draw lines and borders (refer to page 125):
 - MODE0...MODE6 GRAPH BORDER BOX ARROWS

Chapter 7 Video File

Converting a Video File to Binary

	Use the vid2bin utility to convert ASCII video files to binary format for use by applications with the JAM library routines. The ASCII video files distributed with JAM have already been complied and their binary versions reside in the config directory.
	To convert an ASCII video file that you have modified or created to a binary file, use the following:
	vid2bin [-pv] [-e ext] video-file
	The square brackets indicate the optional command flags. Do not type the brackets.
	To obtain a brief description of available arguments and command options, type
	vid2bin -h
	vid2bin searches for the <i>video-file</i> you supply; first trying the mnemonic, then the mnemonic followed by vid. The output file will have the same name as the input file, with the extension bin or the given extension (<i>ext</i>) specified with the –e option.
memory-resident	To make a video file memory-resident, run the bin2c utility (refer to page 563 in the <i>Application Development Guide</i>) on the binary output, compile the resulting program source file, and link it with your application. For a complete description of how to make configuration files memory-resident, refer to page 523 of the <i>Application Development Guide</i> .
	The vid2bin utility checks for errors like missing, misspelled, and superfluous keywords, but not for duplicated or conflicting entries. If errors are encountered during the conversion, up to ten error messages can be displayed; no output file is created.
Arguments and Options	video-file The name of an ASCII video file. Customarily, it is an abbreviation (mnemonic) of the terminal name followed by the suffix vid; for example sunvid for a terminal, or colvid for a color monitor.
	-p Places the binary output file in same directory as input file.
	-v Lists the name of each video file as it is converted.
	-e Appends the given <i>ext</i> (extension) to the output file, rather than the default bin extension.
Errors	The following list describes possible errors, their causes, and the corrective action to take:
104	JAM 7.0 Configuration Guide

A cursor positioning sequence is required.

An erase display sequence is required.

Cause: Both entries (CUP and ED) are required in video files.

Action: Determine what your terminal uses to perform these two operations, and enter them in the ASCII video file; then run vid2bin again.

Invalid entry: '%s'.

Entry missing '=': '%s'.

- Cause: An input line does not begin with a valid video keyword. An input line does not include and an equal sign.
- Action: Correct the ASCII file and run vid2bin again. Be sure that backslashes are placed at the end of lines that continue onto the next line.

Invalid attribute list : '%s'.

Invalid border information (%s):'%s'. Invalid box information (%s): 's%' Invalid color specification : '%s'. Invalid cursor flags specification : '%s'. Invalid graphics character specification (%s):'%s'. Invalid graphics type : '%s'. Invalid label parameter : '%s'.%s Invalid numeric parameter : '%s'

Cause: There is a misspelled or misplaced keyword in the specified input line. Action: Correct the ASCII video file, and run vid2bin again.

Neither %s nor %s found.

Cause: The video file was missing or unreadable. Action: Check the spelling, presence, and permissions of the file in question.

Unable to allocate memory.

Cause: vid2bin could not allocate enough memory for its needs. Action: None.

Video File Keywords

Table 17 includes a list of all video file entry keywords, arranged by function. Detailed information on each keyword is available on the indicated pages. Refer to the sample video file for syntax examples.

Chapter 7 Video File

Table 17. Video file keywords

Keyword	Description
Basic Capabilitie	es (page 109)
BOTTOMRT	Last position of screen may be written without scrolling the display
BUFSIZ	Number of characters to accumulate before flushing
COLMS	Number of columns on screen
INIT	Initialization sequence
KBD_DELAY	Timing interval for keyboard input
LINES	Number of lines on screen
REPMAX	Maximum number of repeated characters
REPT	Repeat the following character sequence
RESET	Undo initialization sequence
Erasure Comma	nds (page 112)
ED	Erase entire display
EL	Erase to end of current line
EW	Erase window
Cursor Position	(page 112)
CMFLGS	Allowed cursor-motion shortcuts
CUB	Cursor backward
CUD	Cursor down
CUF	Cursor forward
CUP	Absolute cursor position
CUU	Cursor up

Keyword	Description		
Cursor Appearance (page 114)			
COF	Turn cursor off		
CON	Turn cursor on		
INSOFF	Overstrike-mode cursor		
INSON	Insert-mode cursor		
RCP	Restore cursor position and attribute		
SCP	Save cursor position and attribute		
Display Attributes (page 115)			
AREAATT	List of available area attributes		
ARGR	Remove area attribute		
ASGR	Set graphics rendition (area)		
COLOR	List of colors		
EMPHASIS_KEEPATT	Specify attributes retained for grayed objects		
EMPHASIS_SETATT	Set attributes for grayed objects		
LATCHATT	List of available latch attributes		
SGR	Set graphics rendition (latch)		
SPXATT	List of attributes that do not affect space		
Status line (page 124)			
CMSG	Close status line		
MSGATT	Status line attributes		
OMSG	Open status line		

Chapter 7 Video File

Keyword	Description
Graphics (page 12	25)
GRAPH	Graphics character equivalents
GRTYPE	Shortcut for defining graphics characters
MODEO	Normal character set sequence
MODE1	Locking shift to alternate character set 1
MODE2	Locking shift to alternate character set 2
MODE 3	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
Borders and Line	Drawing (page 128)
BORDER	Characters that make up the 10 border styles
BOX	Characters that make up the styles for box and line draw ing
BRDATT	Available border attributes
Indicators (page 1	130)
ARROWS	Indicator characters for shifting and scrolling
BELL	"Visible bell" alarm sequence
CBDSEL	Deselection character for groups
CBSEL	Selection character for groups
MARKCHAR	Character used for check menu items
SUBMNSTRING	String on menu item indicating presence of submenu
Drivers (page 132	
BLKDRVR	Name of block mode driver
MOUSEDRIVER	Name of mouse driver

JAM 7.0 Configuration Guide

Keyword	Description
Miscellaneous (page	132)
COMPRESS	Output data compression for Jterm
CURPOS	Display the current cursor position on the status line

Basic Capabilities

BOTTOMRT

A flag indicating that the bottom right-hand corner of the display can be written to without causing the display to scroll. If this flag is not present, JAM does not write to that position.

BUFSIZ

Sets the size of the output buffer used by JAM. If it is omitted, JAM calculates a reasonable default size. Include this entry only if special circumstances warrant. For example, if you make extensive use of a screen-oriented debugger, you can set BUFSIZ to a large value; that effectively disables the delayed-write feature, which can interfere with debugging.

COLMS

refer to LINES Indicates the number of columns on the display. The default value is 80. In some windowing environments (e.g., SUN work stations) the values of LINES and COLMS are overridden by the number of lines and columns in the active window.

INIT

Terminal initialization sequence, output by the library function sm_initcrt. There is no default; and the keyword can be omitted. INIT 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 %#w.

specifying cursor style On MS-DOS systems only, you can use the INIT and RESET sequences (which are normally not used) to specify cursor style. On ASCII terminals, you can use escape

Chapter 7 Video File

sequences for specifying the cursor style in the INIT and RESET strings in the normal fashion. The format is

INIT = C n1, n2[, n3, flag]RESET = C n1, n2[, n3]

n1 and *n2* specify the top and bottom scan lines respectively for the cursor block; with line 0 at the top. (A *scan line* is the smallest vertical unit on your display— one pixel wide). The optional *n3* specifies 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

For a monochrome monitor:	INIT = C 0, 13, 0
For a CGA monitor:	INIT = C 0, 7, 0
For a EGA monitor:	INIT = C 0, 13, 2

In addition, there are flags (listed in Table 18) that you can include the INIT sequence in video files supporting MS-DOS.

Table 18. Optional flags used with INIT on MS-DOS

Flag	Description
	2 000 mp = 000
BIOS	Specifies that JAM should use BIOS calls to display output rather than writing to the video RAM directly.
WINDOWS	Relates to Microsoft Windows. Specifies that JAM can move the cursor, even if the cursor is invisible.
XKEY	Directs JAM to use a different BIOS interrupt for keyboard input that recognizes the F11 and F12 keys on an extended keyboard.
GRAYKEYS	Distinguishes between gray and white cursor positioning keys. For example, this option allows you to assign one value to the gray up arrow key and another value to the white up arrow key.
MULTISHIFT	Permits the use of key sequences using the combinations: Ctrl-Alt, Shift-Alt, and Ctrl-Shift-Alt

KBD_DELAY

Assigns a timing interval to keyboard input. A positive integer between 1 and 10 represents an interval in tenths of a second. A negative integer or 0 represents an interval of indefinitely great length.

If you use key sequences that are lead-ins of other sequences, you must assign a timing interval via KBD_DELAY to determine when a key sequence ends.

LINES

Indicates the number of lines on the display. The default value is 24. In general one line is reserved for status and error messages so the maximum screen size is usually one less than the number specified here. (Refer to OMSG, on page 124, for exceptions.)

REPMAX

Indicates the maximum number of characters REPT can repeat. To determine the proper value of REPMAX, omit the entry from the video file; then using jamdev, create a field that extends the entire width of the screen. Once you choose XMIT, if the entire field has the underline attribute, you do not need the REPMAX entry. If the field is not underlined, gradually shorten the field until the underlines fill the field. The resulting number determines the largest possible value of REPMAX.

REPT

A repeat-character sequence that is passed two parameters: the character to be repeated and the number of times to display it. There is no default, since most terminals do not support character repeat. If it is available on your terminal, include the REPT entry. The repeat sequence is used whenever possible, usually for borders and for clearing areas of the screen. If borders do not appear correctly, your sequence could be incorrect. A repeat sequence is not used to repeat a control character, and it never extends to more than one line.

For example, the following entry outputs the character, then ESC F and the count with 0x3f (the ASCII value of '?') added:

REPT= %c ESC F %+ ?

The next entry sets the maximum count for the preceding REPT entry. Anything over this count splits into more sequences.

REPMAX= 64

RESET

A reset-terminal sequence, output by the library function sm_resetcrt. There is no default. The RESET sequence should be set to undo the effects of INIT. For

Chapter 7 Video File

many terminals a hard reset that resets the terminal to the state stored in non-volatile memory is appropriate.

In video files used on MS-DOS systems, if INIT is specified and RESET is not, JAM saves and restores the original cursor style.

Screen and Line Erasure

ED

Provides the control sequence that erases the display. This entry is required and clears all available display attributes, including background color. You can find the correct command in your terminal manual or in termcap as cl. Some terminals require padding after this command. The following example is common for ANSI terminals

ED = ESC [J

EL

Provides a sequence that erases characters and attributes from the cursor to the end of the line. If EL is not in the video file, JAM erases the line by writing blanks. You can find the sequence in termcap as ce. Some terminals require padding after this command. The first example is common for ANSI terminals; the second illustrates a padded entry:

EL = ESC [K EL = ESC [0 K %100z

EW

Provides a sequence that erases a rectangular region on the screen, to a given background color if available. PCs using MS-DOS use this entry. Five parameters are passed: start line, start column, number of lines, number of columns, and background color. (If color is not available, the fifth parameter is ignored.)

Cursor Position

CMFLGS

Lists shortcuts JAM uses for cursor positioning; they are:

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 $(0 \times 08, \text{ or }^H)$ moves the cursor one position to the left without erasing anything.
- AM Automatic margin: the cursor automatically wraps to the first column when it reaches the right-hand edge of the display.

Note: The AM setting of CMFLGS must match the auto-wrap setting of the terminal. If the setting is not correct, the terminal display may be irregular. Consider using INIT and RESET to turn terminal auto-wrap on and off as desired.

Most terminals are capable of the first three. The fourth can frequently be found in termcap as am. It cannot be used on terminals with the xen1 glitch (i.e., VT100-style delayed auto margin).

CUB

Moves the cursor backward in the same row. Takes the number of columns to move as a parameter. Do not specify this keyword if the sequence can only move the cursor one column at a time. The following entry moves the cursor back:

CUB = ESC [%d D

CUD

Moves the cursor down in the same column. Takes the number of lines to move as a parameter. Do not specify this keyword if the sequence can only move the cursor one line at a time. The following entry moves the cursor down:

CUD = ESC [%d B

CUF

Moves the cursor forward in the same row. Takes the number of columns to move as a parameter. Do not specify this keyword if the sequence can only move the cursor one column at a time. The following entry moves the cursor forward:

CUF = ESC [% C

CUU

Moves the cursor up in the same column. Takes the number of lines to move as a parameter. Do not specify this keyword if the sequence can only move the cursor one line at a time. The following entry moves the cursor up:

CUU = ESC [%d A

Chapter 7 Video File

CUP

Establishes absolute cursor position which is required to run JAM. 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) is used to convert them to be relative to one. 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. The following example is for an ANSI terminal:

CUP = ESC [%i %d;%d H

Another common scheme is to output the line and column as characters, after adding SP. The entry might look like the following example:

CUP = FS C %+SP %+SP

Cursor Appearance

COF

Turns the cursor off. If possible, both COF and CON should be specified. Menus (using a block cursor) look better with the regular cursor off. Also JAM 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.

CON

Turns the cursor on in the desired style. A blinking block cursor is recommended since an underline cursor is difficult to see in an underlined field.

Note: You can use the INIT and RESET sequences to switch between the cursor style used in JAM applications and that used on the command line.

Many terminals have no ability to turn the cursor on and off. Although JAM 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. The following entries are an example of a pair of COF and CON sequences for some ANSI terminals:

 $CON = ESC [>51 \\ COF = ESC [>5h$

INSOFF and INSON

Used with INSON, changes the cursor style so that you can easily see which mode you are in: insert or overstrike. By convention, the insert cursor is about one-half

the size of the regular, overstrike cursor. INSOFF or INSON is issued to the terminal when you toggle JAM's data entry mode using the INSERT key. On many terminals, changing the cursor style also turns it on; in this case, the INSOFF is the same as COF, so you can omit it altogether. If the cursor style can be changed without turning it on or off, use both INSON and INSOFF. Uses the same escape sequence format as INIT and RESET.

RCP and SCP

Saves (SCP) and restores (RCP) cursor position and attribute.

Display Attributes

JAM supports highlight, blink, underline and reverse video attributes. If either highlight or blink is not available, low intensity is supported in its place. An additional attribute, standout, can be assigned to any other desired attribute, e.g. dim or italics, if available. The display attribute keywords AREAATT and LATCHATT are used to list the attributes available in each style and associate a character with each attribute.

Attribute Types JAM supports three different kinds of attribute handling.

- latch attributes Assigns attributes to any characters written after the current cursor position. Latch attributes require no space on the screen. ANSI terminals use this method.
- area attributes Assigns attributes to all characters from the cursor position to the next attribute (or end of line or end of screen). Area attributes do not occupy a screen position (they are "non-embedded" or "no space"). In this style, JAM positions the cursor to the end of the area to be changed, sets the ending attribute, then positions the cursor to the beginning of the area and sets its attribute.
- onscreen attributes Act like area attributes, but occupy a screen position. (They are "embedded" or "spacing.") This style of attribute handling dictates 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 onscreen attributes.

You can set several modes on your terminal. Many terminals support both area and onscreen attributes. If so, you should select area ("non-embedded" or "no space") rather than onscreen ("embedded" or "spacing") attributes. Some terminals support one latch attribute and several area attributes simultaneously.

If your terminal has only one attribute style available, it is recommended that you select reverse video. JAM supports non-display in software, so you can omit that

Chapter 7 Video File

attribute. Underlines are simulated (by writing an underscore character) if that attribute is not available.

You can find attribute information in your terminal's documentation or, perhaps, in your termcap database (if applicable). The codes so, ul and bl specify standout (usually reverse video), underline, and bold respectively. The codes se, ue and be specify the sequence to end the attributes. The standard ANSI sequences are:

 $\texttt{so=} E[7m:\texttt{se=} E[0m:\texttt{ul=} E[4m:\texttt{ue=} E[0m:\texttt{bl=} E[1m:\texttt{be=} E[1m:\texttt{be$

If you find something like these, ANSI latch attributes are available. If you find entries ug#1:sg#1, onscreen attributes are available.

AREAATT

Lists the area or onscreen attributes that are available, and associates a character with each. The possible attributes are:

ACS	Alternate character set (line drawing graphics)
BLINK	Blink or other standout
DIM	Dim (low intensity)
HILIGHT	Highlight (bold)
REVERSE	Reverse (or inverse) video
STANDOUT	User selected standout mode
UNDERLN	Underline

In addition, flags are available that specify how the attributes are implemented by the terminal. The flags are:

LINEWRAP	The attribute wraps from line to line
ONSCREEN	The attribute uses a screen position
REWRITE	Must rewrite attribute when writing character
SCREENWRAP	The attribute wraps from bottom of screen to top

Area and onscreen attributes modify all characters from the start attribute to the next attribute or to an 'end', whichever is closer. (An 'end' is either the end of the screen or the end of the line.) 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 you to select the style. For onscreen attributes, SCREENWRAP is best and LINEWRAP a good second; for area attributes the choices are about the same. If the attribute takes up a screen position, use the ONSCREEN flag.

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

On some terminals 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 disappears. In this case, use the REWRITE flag to reset the attribute before writing to that position. The following example illustrates how the REWRITE flag is used (in Televideo 925 video file):

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

Some terminals restrict the number of attributes, use an ARGR entry to remove attributes.

ARGR

Removes area attributes. Some terminals restrict the number of attributes (as set with ASGR) that are available on a given line. If possible, use an ARGR entry. Changing an attribute to normal does not remove it; a normal attribute stops the propagation of a previous attribute only. The following example illustrates how the ARGR entry might appear in your video file:

```
AREAATT = REVERSE = Q UNDERLN = '
ASGR = ESC d u  %'@' %5( % | %) %c
ARGR = ESC e
```

JAM uses the ARGR entry 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; it sometimes makes the screen "wiggle."

If the maximum number of attributes is small, JAM's performance can be limited. ARGR is desirable because having many attributes onscreen can dramatically slow performance, since JAM must keep rewriting them as attributes change.

Chapter 7 Video File

ASGR

An area set graphics rendition sequence, used in conjunction with AREAATT, is passed twelve parameters. The first nine are the same as used by terminfo. The parameters, in order, represent:

- 1 standout
- 2 underline
- 3 reverse video
- 4 blink
- 5 dim (low intensity)
- 6 highlight (bold)
- 7 blank
- 8 protect not used, always 0
- 9 alternate character
- 10 foreground color (if available)
- 11 background color (if available)
- 12 background highlight

If an attribute is desired, the parameter passed is the character associated with the attribute, as explained in the description of SGR. If the attribute is not desired, the parameter passed is (binary) 0.

If no attributes are specified in the video file, JAM supports only two attributes: non-display (done in software anyway) and underline (using the underscore character).

COLOR

Used to associate a character with each color, just as LATCHATT associates a character with each attribute. JAM supports eight foreground and background colors. The CTYPE entry has flags that tell JAM what background color is available. You only need to specify the three primary colors in the video file. If other colors are not specified, they are generated according to the following rules:

BLACK =	RED & GREEN & BLUE
BLUE	Must be specified
GREEN	Must be specified
CYAN =	GREEN BLUE
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 represents the background color (it is 0 if background color is not available). Many ANSI terminals set foreground color with the following 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, JAM prefers that REVERSE is not specified in the video file. It uses the specified color as the background, and either black or white as the foreground. The following example is 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
```

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:

```
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
```

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

Some terminals use ESC [2 ; x ; y m to set color and other attributes — 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

EMPHASIS_KEEPATT

Use this entry to specify the attributes to be retained when implementing drop shadows and grayed objects. By default, all attributes are enabled except HILIGHT.

Chapter 7 Video File

This variable is used in conjunction with the EMPHASIS setup variable. You can change this setting at runtime with the library function m_pset (refer to the *Language Reference*). Refer to Table 2 on page 21 for a list of display attribute keywords.

EMPHASIS_SETATT

Use this entry to specify the attributes to apply when implementing drop shadows and grayed objects. By default, this variable has two attributes enabled: REVERSE and DIM. This variable is used in conjunction with the EMPHASIS setup variable. You can change attributes at runtime with the library function sm_pset (refer to the *Language Reference*). Refer to Table 2 on page 21 for a list of display attribute keywords.

LATCHATT

Lists the available attributes, and associates a character with each. The possible attributes are:

ACS	Alternate character set (line drawing graphics)
B_HIGLIGHT	Background highlight
BLANK	Non-display (foreground not shown)
BLINK	Blink or other standout
DIM	Dim (low intensity)
HILIGHT	Highlight (bold)
REVERSE	Reverse (or inverse) video
STANDOUT	User selected standout mode
UNDERLN	Underline

The format is:

LATCHATT = attribute = value attribute = value ...

So a typical LATCHATT might look like:

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

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. To determine which attributes are supported and how attributes can be combined, if at all, examine the SGR entry that usually follows the LATCHATT entry.. 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
                                               for normal
                       where p n = 0
                                    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 the normal attribute is
                      wanted, all parameters are 0, and the output sequence is ESC [ 0 m. For underline
                      SGR is ESC [ 0 ; 4 m. If highlighted, blinking, and reverse video are desired,
                      the output is
                      ESC [ 0; 7 ; 5 ;1 m.
unable to combine
                      Some terminals (or emulators) do not accept the method of combining attributes
attributes
                      used above. In that case, one sequence followed by the next might work, e.g., ESC
                      [ 1 m ESC [ 7m. Some terminals cannot combine attributes at all. Here are some
                      more ANSI and near-ANSI examples:
                      "standard" ANSI terminal
                              LATCHATT= HILIGHT=1 BLINK=5 UNDERLN=4 REVERSE=7
                      ANSI with low intensity but no highlight
                              LATCHATT= DIM=2 REVERSE=7 UNDERLN=4 BLINK=5
                      only one attribute available
                              LATCHATT= REVERSE=7
                      repeat of above SGR example
                              SGR = ESC [ 0 %9(%t ; %c %; %) m
                      attributes cannot be combined
                              SGR = ESC [ 0 m %9(%t ESC [ %c m %; %)
                      skip parameters that are always 0
                               SGR = %u ESC [ 0 %5(%t ; %c %; %) m
```

Chapter 7 Video File

In the next LATCHATT/SGR example explicit pushes are used 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" 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
```

bit-mapped attributes Some terminals use bit-mapped attributes. Terminal manuals are not usually explicit on this. Often they use tables like that described in Table 19:

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

Table 19. Bit-mapped attributes

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:



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

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

for Videotex terminal Following are examples of LATCHATT entries that can be used with a Videotex terminal. The LATCHATT entries, when used with the SGR entry, have equivalent equivalent results. The bits are OR-ed 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

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

protected modes Some terminals that use area attributes 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 (%;

SGR

A set graphics rendition sequence, in conjunction with the LATCHATT sequence, is passed twelve parameters.

- 1 standout
- 2 underline
- 3 reverse video
- 4 blink
- 5 dim (low intensity)
- 6 highlight (bold)
- 7 blank
- 8 protect not used, always 0
- 9 alternate character
- 10 foreground color (if available)
- 11 background color (if available)
- 12 background highlight

Chapter 7 Video File

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.

For example, 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 is passed (0, '4', 0, 0, 0, 0, '1', 0, 0, 0). The second and sixth parameters represent underline and highlight; they are set to the corresponding values in the LATCHATT entry. The rest are zero. To make the field reverse video and blinking, SGR is passed (0, 0, '7', '5', 0, 0, 0, 0, 0, 0).

If no attributes are specified in the video file, JAM supports only two attributes: non-display (done in software anyway) and underline (using the underscore character).

SPXATT

Lists attributes which do not change or affect the appearance of a character cell containing a space. For example,

SPXATT = BOLD DIM BLANK BLINK COLOR

For efficiency, this entry reduces the number of characters sent to a screen. It defaults to COLOR BLANK HILIGHT DIM. To turn it off entirely, use

SPXATT =

Status Line

JAM usually uses a line from the screen to display status text and error messages. Thus a 25-line screen (as specified in the LINES keyword) has 24 lines for the screen, and one for messages. This use of a normal screen line for messages is the default.

Terminals that use a separate status line can use different attributes on the status line than on the screen itself. JAM provides some support for this ability; for very complicated status lines, you must write a routine and install it with sm_install using the STAT_FUNC function type. For more information on installing a status line function, refer to page 188 of the *Application Development Guide*.

OMSG and CMSG

Opens and closes the status line. Use the OMSG entry to open the status line, and CMSG to close it. These entries are used primarily for those terminals that have a

special status line that cannot be addressed by normal cursor positioning. All text between these sequences appears on the status line. No assumption is made about clearing the line; JAM always writes blanks to the end of the line.

OMSG = ESC fCMSG = CR ESC g

If the OMSG entry is present in your video file, JAM uses all the lines specified in the LINES entry for screens.

MSGATT

Lists the attributes available on the status line. This keyword takes a list of flags:

AREAATT	All area attributes can be used
BLINK	Blink available
DIM	Dim (low intensity) available
HILIGHT	Highlight (bold) available
LATCHATT	All latch attributes can be used
NONE	No attributes on status line
ONSCREEN	Area attributes take a screen position
REVERSE	Reverse video available
UNDERLN	Underline available

The attribute for the status line is specified as either a latch (LATCHATT) or area (AREAATT) attribute, and the sequence to set it must is given in the SGR or ASGR keyword. For example, if REVERSE is listed in MSGATT and REVERSE is a latch attribute, then SGR sets it. Attributes that appear in MSGATT and do not appear in either LATCHATT or AREAATT are ignored.

In order for JAM to determine the correct length of a line, 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. Use the ONSCREEN flag in MSGATT to inform JAM 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 status line and they take a screen position. The area attributes in the normal screen area do not.

Graphics and International Character Support

JAM 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.

Chapter 7 Video File

GRAPH

Maps internal numbers to output sequences, similar to the key translation files which provide mapping from character sequences to internal numbers. 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 $0 \times a0$ to $0 \times ff$. JAM 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 key translation file can be used to map such sequences to single values in the range $0 \times a 0$ to $0 \times f \in$. (Refer to the *Application Development Guide* for a way to use values outside that range.) The video file can 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 entries tell which alternate set to use for each graphics character, and how to shift to it. Whenever JAM is required to display a character, it looks in the GRAPH table for that character (refer to the description on MODE0 through MODE6 for information on what happens). If it is not there, the character is sent to the terminal unchanged.

GRTYPE

Provides a convenient shortcut for certain common graphics sets, each denoted by another keyword. The GRTYPE keywords may be combined.

The format is GRTYPE = [GRTYPE] keyword(s)

The GRTYPE *keyword(s)* are:

ALL	0xa0 through 0xfe
EXTENDED	same as ALL
PC	0x01 through <code>0x1f</code> and <code>0x80</code> through <code>0xff</code>
CONTROL	0x01 through 0x1f, and 0x7f
C1	0x80 through 0x9f, plus 0xff

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.

MODE0 through MODE6

JAM supports up to three alternate character sets. The sequences that switch among character sets are listed below.

MODES0 through MODE3 are locking shifts. All characters following are shifted, until a different shift sequence is sent.

MODES4 through MODE6 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.

MODEO	switch to standard character set
MODE1	alternate set 1
MODE2	alternate set 2
MODE 3	alternate set 3
MODE4	
MODE5	
MODE6	

Different modes can be used to support foreign characters, currency symbols, graphics, etc. JAM 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, JAM 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:

```
MODE0 = SI

MODE1 = SO

MODE2 = ESC n

MODE3 = ESC o

sample of ANSI

MODE0 = ESC [ 10 m

MODE1 = ESC [ 11 m

MODE2 = ESC [ 12 m

MODE3 = ESC [ 13 m

MODE0 =

MODE1 = SS1

MODE2 = SS2

Any of the MODEn strings m

that mode is displayed, that
```

Any of the MODEn strings may also contain a list of attributes. When a character in that mode is displayed, that attribute is added to whatever attribute is already in effect. On some terminals, like the HP, only an attribute is required. For example

MODE4 = ACS

which forces all mode 4 characters to be displayed using the alternate character set.

Chapter 7 Video File

Any character in the range 0×01 to $0 \times ff$ 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 $0 \times 90 = 1$ d appears in the GRAPH list. First the sequence listed for MODE1 is 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 JAM. 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 rings the terminal bell.

	MODEO = SI
	MODE1 = SO
	MODE2 = ESC n
	$MODE3 = ESC \circ$
	GRAPH = 0x81 = 1 / 0xb2 = 1 2 0x82 = 3 a LF = 0 SP
	BEL = 0 SP CR = 0 SP
commendations	For efficiency, 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 = 0 z causes the y key to display as z. Graphics characters are non-portable across different displays, unless care is taken to ensure 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.

Borders and Line Drawing

The characters constituting the border and line drawing styles can be specified in the video file.

BORDER

Specifies alternate borders. If fewer than 10 are given, the default borders (listed below) are used to complete the set. They are numbered 0 to 9. When you create screens with the screen editor, you can select from these border styles. The BORDER entry is portable across different platforms because JAM saves a border as a style number in the screen file. Style 1 is the default.

0.	1.
IIIII	
II	
IIIII	

JAM 7.0 Configuration Guide

re

2.	3.
+++++	=====
+ +	
+++++	=====
4.	5.

8 8	: :
8 8 8 8 8	
б.	7.
* * * * *	\\\\\
* *	\ \ \
* * * * *	
8.	9.
////	#####
/ /	# #
	#####
/////	πππππ

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	\backslash							
		SP	_	SP				_		\backslash
		+	+	+	+	+	+	+	+	\backslash
		SP	=	SP			SP	=	SP	\backslash
		%	%	00	00	00	00	%	00	\backslash
				•	:	:	:		:	\backslash
		*	*	*	*	*	*	*	*	\backslash
		\backslash								
		/	/	/	/	/	/	/	/	\backslash
		#	#	#	#	#	#	#	#	
The following example is for the PC graphics character set:										

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

In addition, attributes can be specified for each set of border characters. For example:

Chapter 7 Video File

BORDER = SP SP ... SP REVERSE \langle ; A ... + ACS \langle

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.

BOX

Ten different sets of line draw characters can be specified when you are using the screen editor. The BOX entry lists either five or thirteen characters per set. If only five characters are specified the remaining eight are taken from the corresponding BORDER set.

Although the format in the video file is similar, JAM uses BOX and BORDER differently. While BORDER entries are portable across platforms, JAM saves line drawing as display data. To ensure portability, avoid assigning graphic characters to the BOX keyword. Instead, use characters that are displayable on all terminals.

BRDATT

Use a BRDATT entry to limit the attributes available in the border. Normally HILIGHT (or DIM) and REVERSE are used; however, if your terminal uses onscreen attributes or can accommodate only a few attributes per line, it may be better to prohibit attributes in borders — use the entry BRDATT = NONE.

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

Indicators

ARROWS

Used to indicate the presence of offscreen data in shifting/scrolling fields. You can define these indicators to be any characters you wish. The default characters are:

<, >, X for shifting

^, v, x for scrolling

The character x is used when two shifting/scrolling fields are next to each other; it represents a combination of both < and >.

ARROWS = $\langle \rangle X ^ v X$

Shift/scroll indicators are black on screens with background colors of white, yellow, or cyan. They are white on all other screen background colors. You cannot alter indicator colors.

BELL

If present, is transmitted by the library function sm_bel to give a visible alarm. Normally, the function rings the terminal's bell. The BELL sequence can sometimes be found in the termcap file under vb.

CBDSEL and **CBSEL**

Selection (CBSEL) and deselection (CBDSEL) characters for groups. If there are no entries for CBSEL and CBDSEL in your video file, the internal defaults are x for CBSEL and a blank for CBDSEL. For radio button or checklist widgets, these characters are used to indicate which fields are selected and which are not. You can add these entries to the video file to override the defaults. For example:

```
CBSEL = y
CBDSEL = n
```

As a result, JAM uses a y to indicate a selected occurrence; NL deselects the occurrence, and JAM inserts an n in the box.

MARKCHAR

Defines the character used to check menu items. The default character is x. For example, the following example specifies the square root symbol ($\sqrt{}$) as the mark character. This variable is optional for supporting those character-mode applications that use menu bars and is most useful in video files that support reasonable graphical characters.

```
MARKCHAR = 0xFB
```

SLIDER

Defines the characters used in character-mode scroll bars. Eight characters are defined, where the first set of four characters define the slider characters used in horizontal scroll bars, and the second set contains the slider characters used in vertical scroll bars:

SLIDER = left-arrow right-arrow bar thumb \ up-arrow down-arrow bar thumb

For example, these characters are defined in vt100vid:

SLIDER = < > = # ^ v / #

You can use hex values of graphics characters for those terminals that support them. For example:

SLIDER = 0x11 0x10 0xb0 0xb2 0x1e 0x1f 0xb0 0xb2

Chapter 7 Video File

SUBMNSTR

Defines the indicator for submenu options. This indicator appears on menu bar options indicating that the item invokes a submenu. The default string is an ellipsis (...). This variable is optional for supporting those character-mode applications that use menu bars and is most useful in video files that support reasonable graphical characters.

Drivers

MOUSEDRIVER

Enables the mouse in JAM applications. Supported mouse types on UNIX are kb, AT, JAMPI, and xterm. On PCs, you can only specify PC. For example, you might add the following entry to your PC video file:

MOUSEDRIVER = PC

Miscellaneous

COMPRESS

Implements data compression for Jterm users. Use the following entry:

COMPRESS = JTERM

CURPOS

Specifies the time-out delay, in tenths of a second. The CURPOS entry causes the current cursor position to be displayed on the status line at the specified intervals:

CURPOS = 1 Updates display every 0.1 second (use on fast systems)

CURPOS = 3 Updates every 0.3 second (reasonable for most)

CURPOS = 7 Updates every 0.7 second at low baud rates

CURPOS = 0 No display, same as omitting keyword.

When possible, JAM uses non-blocking keyboard reads. If no key is received 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 delay depends on the baud rate and your terminal. If there is no non-blocking read, a non-zero value of CURPOS enables the display and zero disables it. If the terminal has its own display, omit CURPOS.
For Basic ANSI Terminal	This sample video file, for a basic ANSI terminal (like a VT100), 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 runtime.
	# Display size (these are actually the default # values) LINES = 24 COLMS = 80
	# Erase whole screen and single line ED = ESC [2 J EL = ESC [0 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 %u %5(%t ; %c %; %) m
For MS-DOS	By default, JAM 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 for these machines. Under no circumstances does JAM use DOS calls or the ANSI.SYS driver. Video files for both mono-chrome and color displays are distributed with JAM.
	Because JAM 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

LINES, GRAPH, and BORDER, can be changed.

distributed video files. Do not alter these entries; they are required but their values are irrelevant. Other entries, that do not contain PC control sequences, such as

Chapter 7 Video File

defines the display area \rightarrow	LINES = 25 COLMS = 80	
	<pre># INIT and RESET can change # other flags are available</pre>	e the cursor style if desired e in the INIT sequence.
cursor style>	<pre># the default for INIT is: # INIT = C 0,7,2</pre>	0 specifies top scan line; 7 specifies bottom scan line; 2 specifies fast blinking cursor.
	<pre># most sequences are handle # these should be set to ""</pre>	ed by assembler functions, PC" to function correctly
erase display > erase to end of line >	ED = PC EL = PC CON = PC <	n f
cursor style in insert mode cursor style in overstrike mode	INSON = C 6,7,0 INSOFF = C 0,7,0	6 specifies top scan line; 7 specifies bottom scan line; 0 specifies rapid blinking cursor restores cursor to block style, blinking cursor.
absolute cursor	$\begin{array}{llllllllllllllllllllllllllllllllllll$	
available colors> available attributes>	COLOR = BLUE = 1 GREEN = 2 LATCHATT = HILIGHT BLINK SGR = PC <	RED = 4 BACKGRND sets attributes specified by LATCHATT
erase window	EW = PC	
	SCP = PC <	save cursor position and attributes
	RCP = PC <	<pre>_ restore cursor position and attributes</pre>
repeat character specification	REPT = PC	

	# # #			 			 	
PC graphics characters specify border style	BORDER = 0xda 0xc9 0xd5 0xd6 0xdc 0xb0 0xb2 0xbd	SP SP 0xc4 0xcd 0xc4 0xc4 0xb0 0xb2 0xb0 0xb2	SP SP 0xbf 0xbb 0xb8 0xb7 0xdc : : 0xb0 0xb2 0xbd	SP SP 0xb3 0xba 0xb3 0xba 0xdd 0xb0 0xb2 0xbd	SP SP 0xb3 0xba 0xb3 0xba 0xba 0xb0 0xb0 0xb2 0xbd	\ 0xc0 0xc8 0xd4 0xd3 0xdf 0xb0 0xb2 0xbd	0xc4 0xcd 0xcd 0xc4 0xdf 0xb0 0xb2 0xbd	0xd9 \ 0xbc \ 0xbe \ 0xbd \ 0xdf \ 0xb0 \ 0xb2 \ 0xbd
PC graphics characters specify box style	<pre># # - BOX = SP 0xc2 0xcb 0xd1 0xd2 0xb1 0xb2 0xb0 0xb2 0xb0 0xb2 0xdb # # <></pre>	- SP SP 0xc3 0xcc 0xc6 0xc7 0xb1 0xf9 0xb0 0xb2 0xb2 0xdb ^ <->	 SP SP 0xc5 0xce 0xd8 0xd7 0xb1 0xf9 0xb0 0xb2 0xb2 0xdb ^ v v	\ 0xb4 0xb9 0xb5 0xb6 0xb1 0xf9 0xb0 0xb2 0xb2 0xdb	0xc1 \ 0xca \ 0xcf \ 0xd0 \ 0xb1 \ 0xb9 \ 0xb0 \ 0xb2 \ 0xdb			
undates cursor position	ARROWS =	0x1b 0x	la 0x1d	0x18 ()x19 Ox	12	specifies ir for shifting,	ndicator styles /scrolling fields
display every .10 sec	CURPOS =	1	specifie	s graphic	Ov01 the	ough Outf		brough Ouff
	GRTYPE =	PC <	- characte	er set				niougn oxii
specifies mouse driver	►MOUSEDRIV	ER=PC	enables	mouse in	ЈАМ аррі	Ications		
	MARKCHAR= SUBMNSTRI	û NG=ÍÍÍ^	P					

Chapter 7 Video File



Configuration Map File

The configuration map file contains definitions for for screens and widgets—colors, fonts, lines and box styles—that you can tailor to different platforms. The file is divided into several sections:

- [Colors] maps user-defined color names to system color names (page 138).
- [Schemes] maps color definitions to application components such as screens and widget types (page 142).
- [Lines] defines line and box styles (page 145).
- A fonts section—[Windows Fonts] for Windows, [Display Fonts] for other GUI platforms—tells JAM which fonts and font sizes to display in the drop-down menus in the screen editor; it also defines default display fonts, and maps system-specific font names to JAM font aliases. (page 147).

By defining these display elements in GUI-specific files and using their names for screen and widget properties, you can create applications that are easy to port across different platforms. Instead of creating multiple instances of the same screens that each use GUI-specific font and color names, you can create multiple configuration map files—one for each platform on which JAM and your JAM applications run. For example, you can create a color alias PanicButtonRed that resolves to different colors in different configuration maps.

JAM is installed with at least one configuration map file (*cmap) that suits your environment. You can edit these or create your own with an ASCII text editor, then run the utility cmap2bin to convert it to binary format (page 150).

During initialization, JAM looks for the configuration variable SMCOLMAP which can be defined in the environment or in an SMVARS file). This variable gives the full pathname of the binary configuration map file.

Defining Colors

When you create a screen or widget, the screen editor seeks default color settings, or a *scheme*, for that object's type. foreground and background. The editor automatically sets the color property to Scheme and then resolves the scheme, looking first in the configuration map file. If the file provides no scheme for the object, the editor looks elsewhere for color defaults (refer to page 142).

JYACC provides configuration maps with default schemes for screens and for each widget type. You can define your own color schemes that suit your environment, style preferences, or development and application requirements. Or you can rely on the local GUI to assign colors to your application objects.

You can set the Color Type property to one of these three settings:

- Scheme—the defaults defined in the [Schemes] section in the *cmap file, or if none, a set of default colors determined by settings defined either in the native GUI or in JAM.
- Basic—JAM's eight highlighted and eight unhighlighted colors, plus the Container option. (The Container option specifies that a widget within another object has the same background color as the container.)
- Extended colors—GUI-specific colors that are specified by a string and are resolved in the [Colors] section of the *cmap file, or directly by the GUI.

Color Aliases

The [Colors] section defines GUI-independent color aliases that you can use in the Color Name property of screens and widgets. All color names, including JAM palette color names like hilight_red, must be added to the list of color aliases. Each entry appears on its own line in the following format:

alias_color = color

alias_color

Any name you choose to identify a color.

color

One of the following:

• An RGB value in a platform-specific form:

For Windows, use the form "*red/green/blue*" where *red*, *green* and *blue* are numbers between 0 and 255. For example:

PanicButtonRed = "205/92/92"

For Motif, use the form "#*RedGreenBlue*" where *Red*, *Green*, and *Blue* are hex numbers between 00 and ff. For example:

PanicButtonRed = "#cd5c5c"

• A GUI specific name (in Motif only). For more information on Motif color names, refer to page 171. For example:

PanicButtonRed = "Indian Red"

JAM keywords in the form basic_color (attributes); where basic_color corresponds to one of JAM's 16 colors (or "container") and attributes is an optional display attribute (refer to Table 20). For example:

NumberField = BLACK UNDERLN

This style of definition can create a GUI-independent color alias. For example:

SpringGreen=Green Hilight SummerGreen=Green Dim

If a widget had the Motif color SpringGreen specified and JAM could not find it, it would substitute the JAM color Green Hilight, which is always defined. A configuration map with similar aliases would allow a Motif-specific screen to appear similarly when running in JAM's character mode.

Note: In Windows and Presentation Manager, JAM screens and widgets that have highlighted background colors are different from those having unhighlighted background colors. In character JAM on a PC under DOS, there is normally no difference between highlighted and unhighlighted background colors. JAM display attributes have no effect in Motif.

Chapter 8 Configuration Map File

Color	Keyword	Attribute	Keyword
Black	BLACK	Reverse video	REVERSE
Blue	BLUE	Underline	UNDERLN
Green	GREEN	Blink	BLINK
Cyan	CYAN	Highlight	HILIGHT
Red	RED	Dim	DIM
Magenta	MAGENTA		
Yellow	YELLOW		
White	WHITE		
Container	CONTAINER		

Table 20. JAM color and attribute keywords. Keywords are case-insensitive.

The keyword CONTAINER specifies that a widget within another object has the same background color as the container. Therefore CONTAINER can not be used to specify a foreground color. Also, because CONTAINER may contain attributes, you cannot specify any additional attributes with it.

Screen Editor Colors A number of predefined color aliases control the screen editor's appearance. All screen editor color aliases begin with se; entries use the same format as user-defined colors. For example, this entry in a Windows or Presentation Manager configuration map file sets the background color of the design screen:

seFormBg = "127/255/0"

Table 21 lists screen editor color aliases and the objects whose appearance they control:

Color alias	Description
seBorderFG	Editor windows border foreground.
seCheckFG	Property window option menu foreground.
seEntryFG	Property window text field foreground.
seFormBG	Editor windows background.
seLabelFG	Label foreground.
seListBG	List background (except for Property window).
seListFG	List foreground (except for Property window).

Table 21. Screen editor object constants. Object keywords are case sensitive.

JAM 7.0 Configuration Guide

Color alias	Description
seMultiBG	Multiline text background.
seMultiFG	Multiline text foreground.
seOptionmenuBG	Option menu background.
sePushBG	Push button background.
sePushFG	Push button foreground.
sePwListBG	Property window list background.
sePwListFG	Property window list foreground.
seTbBorderFG	Tool box border foreground.
seTbFormBG	Tool box background.
seTbTogFG	Tool box toggle button foreground.
seTextBG	Text background.

Sample Colors Section The following examples are from ASCII configuration map files; the aliases ensure that colors you specify for one platform are displayed correctly on others without editing Color Name properties of application components. Given the appropriate configuration map file, an application displays colors that are correct for its environment.

The [Colors] section in the Motif configuration map defines these color aliases:

Slate Gray	=	"#708090 <i>"</i>
Olive Drab	=	"#6B8E23″
ButtonBlue	=	"#0938EE"

For character mode, the [Colors] section redefines these aliases with JAM color names:

Slate Gray	=	HILIGHT	WHITE
Olive Drab	=	Green	
ButtonBlue	=	Blue	

For a Windows and Presentation Manager configuration map, these aliases are redefined with RGB values:

Slate Gray	=	"112/128/144"
Olive Drab	=	"107/142/35"
ButtonBlue	=	"09/38/240"

If you specify Slate Gray on the three platforms, the correct color is displayed. If you alias the Motif color to map to a JAM-specific color, you ensure that when

Chapter 8 Configuration Map File

your application runs in character JAM, Slate Gray is displayed as the JAM color hilight white.

Color Schemes

	You can decide on a set of default colors for each newly created object in the JAM screen editor. When the Color Type property is set to Scheme, JAM uses the configuration map file to resolve the object's foreground and background colors, according to its type.
	The Schemes section of the configuration map file can include explicit settings or defer to the GUI's resource database or initialization file.
Default Schemes	If the configuration map file omits a [Schemes] section, JAM uses the following default schemes:
	• Character JAM: white foreground, black background.
	• Motif: the *fg and *bg settings in the resource database.
	• Windows: Control Panel colors.
	• Presentation Manager: Color Palette colors.
Scheme Syntax	Each entry in the [Schemes] section appears on its own line in the following format:
	object = color
	object Any widget type, including lines and boxes, screen, and borders, followed by either a foreground (FG) or background (BG) mnemonic; for example, ToggleBut- tonFG and ListBoxBG. Refer to Table 22 for a list of valid object specifications.
	color

One of the following specifications:

• An RGB value in a platform-specific form:

For Windows and Presentation Manager, use the form "red/green/blue" where red, green and blue are numbers between 0 and 255. For example:

MultiTextFG = "0/0/255"

For Motif, use the form "#*RedGreenBlue*" where *Red*, *Green*, and *Blue* are hex numbers between 00 and ff. For example:

MultiTextFG = "#0000ff"

○ JAM keywords in the form basic_color [attribute]; where basic_color corresponds to one of JAM's 16 colors (or "container") and attribute is an optional display attribute. (Refer to Table 20.) (You may not use the container color for foreground color designations.) For example:

TEXTFG = BLACK UNDERLN

• A GUI independent color alias. For example, this entry sets LabelBg to JYACC blue, an alias that must be defined in the [Colors] section:

LabelBG=JYACC blue

• Use the keyword GUI to indicate the native GUI resource database or initialization file. For example, the following indicates that the native GUI resolves the foreground color for toggle buttons:

TogglebuttonFG=GUI

• Use the keyword GUI and the Motif or Windows/Presentation Manager screen element scheme value. (Do not use any additional attributes with a GUI keyword color designation.) For example, for Windows/Presentation Manager:

PushButtonFG = GUI Buttonface

For Motif:

PushButtonBG = GUI XJam*background

 GUI-specific colors. These exist only in Motif, and used un-aliased, limit the scheme's color portability to other environments. For more information on Motif color names, refer to page 171.

CheckBoxFg =tomato

Table 22. Object specifications for setting schemes. Object specification keywords are case-insensitive.

Object Specification	Descriptions
BoxTopBg	Top border background of box widget.
BoxTopFg	Top border foreground of box widget.
CheckBoxBg	Check box background.
CheckBoxFg	Check box foreground.
ComboBoxBg	Combo box background.
ComboBoxFg	Combo box foreground.

Chapter 8 Configuration Map File

Defining Colors

Object Specification	Descriptions
FormBg	Screen color scheme.
FormBorderBg	Screen border background.
FormBorderFg	Screen border foreground.
GraphBg	Graph widget background.
GraphFg	Graph widget foreground.
GridBg	Grid widget background.
GridFg	Grid widget foreground.
LabelBg	Static label background.
LabelFg	Static label foreground.
LineBg	Line widget background.
LineFg	Line widget foreground.
ListBoxBg	List box background.
ListBoxFg	List box foreground.
MultiTextBg	Multitext background.
MultiTextFg	Multitext foreground.
OptionMenuBg	Option menu background.
OptionMenuFg	Option menu foreground.
OutputBg	Dynamic label background.
OutputFg	Dynamic label foreground.
PushButtonBg	Push button background.
PushButtonFg	Push button foreground.
RadioButtonBg	Radio button background.
RadioButtonFg	Radio button foreground.
ScaleBg	Scale widget background.
ScaleFg	Scale widget foreground.
TextBg	Single line text background.
TextFg	Single line text foreground.
ToggleButtonBg	Toggle button background.
ToggleButtonFg	Toggle button foreground.

Defining Line and Box Styles

[Lines] section entries map character-mode styles for lines and boxes to GUI styles. Character-mode line and box styles are defined in the box and border entries of your terminal's video file.

Each entry appears on its own line in the following format:

style name = style content

style name

A predefined or new style name. Spaces are allowed and case is irrelevant.

style content

A predefined style name or another alias style name defined in this file. Spaces are allowed and case is irrelevant. Currently supported predefined style names include:

Dash	Dashdot	Dashdotdot	Default
Dot	Double Dash	Double	Etched In
Etched In Dash	Etched Out	Etched Out Dash	In
None	Out	Single	
Style 0	Style 1		Style 9

You can use this section of the configuration map file to assign the styles 0 through 9 to GUI-specific line styles. For example, you might define the following entries for Motif:

[Lines] style 0 = etched in style 1 = etched out

These entries tell JAM for Motif that when to interpret style 0 as an alias for etched in, and style 1 as etched out.

Character Mode

Styles 0 through style 9 are native to JAM running in character mode. Style 1 is defined as the default line style. When you assign a character–specific style as the Style property value for a line or box style in the screen editor, that style is mapped

Chapter 8 Configuration Map File

to style 1 on non-character JAM applications. GUI-specific styles map to style 1 when running in character mode.

GUI Styles

The default line and box style for all GUI platforms is etched in. Table 23 shows which styles are supported by different platforms, and how JAM displays styles that are undefined or are not supported by the GUI. Supported styles are represented by asterisks (*). Because Windows and Presentation manager support the same styles for lines and boxes, the table does not differentiate between these two widgets; however, Motif supports a different set of styles for each widget type, so these are depicted separately.

Note: In Windows and Presentation Manager, screens that have their 3D property set to No display Etched In and Etched Out as single lines.

Table 23. Mapping of JAM line and box styles on GUI platforms.

Line styles	Windows/PM	Motif line	Motif box
Default	etched in	etched in	etched in
Style 0	single	no line	etched in
None	single	no line	etched in
Styles 1–9	single	etched in	etched in
Etched In	*	*	*
Etched In Dash	dash	*	etched in
Etched Out	*	*	*
Etched Out Dash	dash	*	etched in
Single	*	*	etched in
Dash	*	*	etched in
Dot	*	dash	etched in
Dashdot	*	dash	etched in
Dashdotdot	*	dash	etched in
In	single	etched in	*
Out	single	etched out	*

* Style is supported by the GUI platform.

Line styles	Windows/PM	Motif line	Motif box
Double	single	*	etched in
Double Dash	single	*	etched in

* Style is supported by the GUI platform.

To control the mapping, assign the desired specification in the configuration map file.

Defining Display Fonts

Display font information is contained in its own section—[Windows Fonts] under Windows, [Display Fonts] for other platforms. Entries in this section let you:

- Specify the fonts and point sizes that appear on drop-down menus for the Font Name and Point Size properties.
- Specify the default font and font size.
- Define font aliases.

Point Sizes

You can specify the point sizes that appear on the Point Size property's drop-down menu with an entry that has this format:

point_sizes = size[size]...

For example:

point_sizes = 8 9 10 12 14 16 18 20 24 36 48 72

Note: JAM uses the point_sizes entry only for scalable fonts. For a non-scalable font, JAM gets its available sizes from the GUI and displays these on the drop-down menu.

Default Font

You can specify the default font that JAM applies when you accept Default for a screen's Font Name property with an entry that has this format:

Chapter 8 Configuration Map File

default_font = *font-spec*

font-spec is a font specification that is valid for this configuration map file's environment. For applications running on Windows, specify the font name only. For example:

```
default_font = Arial
```

For Motif applications, specify fonts with the XLFD font naming convention; substitute the wild card character * for all weight, slant, and size properties. For example:

```
default_font = -*-Helvetica-*
```

Note: JAM also uses the default font for any font alias that it cannot resolve.

Default Font Size

The default_font_size entry specifies the font size that JAM applies when you accept Default for a screen's Font Size property. Use this format:

```
default_font_size = size
```

Font Aliases

The fonts section can define any number of GUI-independent font aliases that appear on the Font Name property's drop-down menu. In GUI environments, JAM merges these with the names of fonts supplied by the GUI itself.

Each font alias definition has the following format:

alias-name [(font-qualifier...)] = font-spec [[(font-qualifier...)] = font-spec]...

You can reiterate the same font alias with different qualifiers on separate lines, and thereby map it to unique font specifications. For example, the following alias definition uses different qualifiers to map font alias Text to two different fonts, depending on whether the Italic property is set:

```
Text (noitalic) = Arial
(italic) = ArialItalic
```

If more than one entry matches a widget's properties, the first matching entry determines which font is displayed.

The following sections discuss each component of a font alias definition.

alias-name

The name that you choose to identify a font.

font-qualifier

Optionally limits usage of *alias-name* to those objects that also use the specified qualifiers. You can AND together one or more space-delimited font qualifiers from each of the following columns, in any order:

bold italic underline *point-size[point-size]...* nobold noitalic nounderline

For example, a Windows configuration map file might contain two definitions for the font alias Helv, the first qualified, the second unqualified:

If a widget's Font Name property is set to Helv, JAM uses Arial unless two other conditions are also true: the Italic property is set to Yes, and the Point Size property is set to either 12 or 14. In this case, JAM uses ArialItalic.

Point size qualifiers can limit the number of choices available in the Point Size's drop-down menu. For example, given the following Windows alias definition, choosing SmallFont as a widget's Font Name property limits the choices on the Point Size drop-down menu to Default, 8, and 10:

SmallFont (8 10) = Arial

Note: Point size qualifiers are used on the Point Size property's drop-down menu only if they are valid for the selected font.

font-spec

font-spec maps the font alias to a font supported by the GUI environment. For applications running on Windows, specify fonts with this syntax:

fontname[-point-size][-bold][-italic][-underline]

For example:

Title = Arial-14-bold Text = Arial

For Motif applications, specify fonts with the XLFD font naming convention:

-foundry-family-weight-slant-width-style-pixel size-point size-x resolutiony resolution-spacing-average width-charset registry-charset encoding

Chapter 8 Configuration Map File

You can substitute any component in an XLFD font name with the wild card character *. For example:

```
Courier = -*-courier-*-r-*
Courier (italic) = -*-courier-*-o-*
```

If *font-spec* omits values for point size, slant, or weight, JAM supplies these values from the corresponding property settings—Point Size, Italic, and Bold. For example, the following entries for font alias Helv—each in separate configuration map files for Windows and Motif—specify only the font's family name:

Helv = Arial Helv = -*-helvetica-*

Given these definitions, any widget using Helv as its font can also have its Point Size, Bold, and Italic properties set; these properties are used to resolve the displayed font. So, if the widget's Bold and Italic properties are set to Yes, JAM resolves the aliases to Arial-bold-italic on Windows and -*-helvetica-bold-i-* in Motif, and passes on these specifications to their respective GUIs.

Conversely, these definitions of font alias HelvBold sets its weight to bold:

```
HelvBold = Arial-bold
HelvBold = -*-helvetica-bold-*
```

The explicit weight specifications for HelvBold override the Bold properties for a widget that uses this font; the font is always displayed as bold.

Converting Configuration Map Files to Binary

You use the cmap2bin utility to convert ASCII configuration map files to a binary format for use by JAM. cmap2bin automatically appends the binary file name with the bin extension (unless the -e option is used). It places the binary output file in the directory from which the utility is run.

To convert a configuration map file, use the following format:

Synopsis cmap2bin [-pv] [-e ext] map file [map file ...]

Options and map file Arguments The nam

The name of the ASCII configuration map file. More than one input file may be specified.

JAM 7.0 Configuration Guide

- -p Output file(s) will be placed in same directory as input files(s).
- -v Generates a list of files processed.
- -e ext Uses ext as the file extension to the output file(s).

Sample Configuration Map File

The following configuration map file defines colors, fonts, and lines styles for Windows applications.

```
[Colors]
                       = MAGENTA
                                               # JAM color
grape
Aquatic Blue = "64/32/200" # Windows-style RGB value
# The following entries in the color map are for use in the
# screen editor. If you remove them entirely, then SCHEME
# colors are used, which may be desirable in Windows.
#seFormBG = GUI WindowBackground
#seBorderFG = Unused by Pi for Windows
#seLabelFG = GUI WindowText
#sePushFG - CUT -
#sePushFG
#sePushBG
                      = GUI ButtonText
#sePushFG = GUI ButtonText
#sePushBG = GUI ButtonFace
#seEntryFG = GUI WindowText
#seMultiFG = GUI WindowText
#sePwListFG = GUI WindowText
#sePwListBG = GUI WindowText
#seListFG = GUI WindowText
#seListBG = GUI WindowText
#seCheckFG = GUI WindowText
#acoPutionmenuPC = CUI WindowText
#seOptionmenuBG = GUI WindowBackground
#seComboboxBG = GUI WindowBackground
#seTextBG = GUI WindowBackground
# The following definitions are for the tool box
seTbFormBG = BLACK
#seTbBorderFG = Unused by Pi for Windows
#seTbTogFG = Unused by Pi for Windows
[Schemes]
#OUTPUTFG = GUI WindowText
#TEXTFG = GUI WindowText
#MULTITEXTFG = GUI WindowText
#PUSHBUTTONFG = GUI ButtonText
#TOGGLEBUTTONFG = GUI ButtonText
#RADIOBUTTONFG = GUI WindowText
```

Chapter 8 Configuration Map File

#OPTIONMENUFG = GUI WindowText #OPTIONMENUFG= GUI WindowText#COMBOBOXFG= GUI WindowText#LISTBOXFG= GUI WindowText#SCALEFG= GUI WindowText#LABELFG= GUI WindowText#BOXTOPFG= GUI WindowText#LINEFG= GUI WindowText#CHECKBOXFG= GUI WindowText = GUI WindowFrame #CHECKBOXFG = GUI Windowiexc #FORMBORDERFG = Unused by Pi for Windows GRAPHFG = BLACK #GRIDFG = GUI WindowText #FORMBG = GUI WindowBackground
OUTPUTBG = CONTAINER
#TEXTBG = CUIT TO #MULTITEXTBG = GUI WindowBackground **#PUSHBUTTONBG** = GUI ButtonFace **#TOGGLEBUTTONBG = GUI ButtonFace** RADIOBUTTONBG = CONTAINER **#OPTIONMENUBG** = GUI WindowBackground #OPTIONMENUEG = GUI WindowBackground #COMBOBOXEG = GUI WindowBackground #LISTBOXEG = GUI WindowBackground SCALEEG = CONTAINER LABELEG = CONTAINER #BOXTOPEG = CONTAINER #LINEEG = Unused by Pi for Windows CHECKBOXEG = CONTAINER #FORMBORDERBG = Unused by Pi for Windows #GRAPHBG = CONTAINER #GRIDBG = CONTAINER [Lines] Style 1 = Single MyFavoriteStyle = Double [Windows Fonts] # Point Size property drop-down point_sizes = 8 9 10 12 13 14 16 18 20 22 24 26 28 36 48 72 # Application defaults for Font Name and Point Size # properties default_font (print) = Times New Roman default_point_size (print) = 10 # JAM Font Name Qualifiers Windows font # ----- ----- ------= Courier New JAM Courier = Times New Roman JAM Times Roman JAM Helvetica = Arial JAM Symbol = Symbol



Setting Windows Defaults

Applications that run under Microsoft Windows use initialization files to set defaults for the GUI. The initialization file controls how Windows and the applications running under Windows appear and behave. You can set up the initial state, and (as is the practice in Windows) your users can change these settings to suit their preferences.

The settings you establish in the initialization file override any duplicate settings you have indicated in your SMVARS files.

Initialization Files

location

The initialization files reside in the Windows directory. Preferences are indicated in the initialization file by setting attribute/value pairs. JAM applications running under Windows use initialization files to determine values for a variety of attributes including:

- Default colors
- Mapping between JAM colors and GUI-specific colors

- GUI independent color names
- Application behavior

Filenames

Each application can have an application-specific initialization file. The name of this file is set by the argument to JAM's GUI initialization function. To change the initialization filename, edit the argument to the function sm_pi_mw_setup in the file piinit.c for a JAM application executable, or the function sm_pi_mw_jxsetup in the file pijxinit.c for a JAM development executable. These source files can be found in the link directory.

At initialization, the main routine of your application (usually either jmain.c or jxmain.c) calls either the function sm_pi_init or sm_pi_jxinit to initialize the GUI. These routines in turn call sm_pi_mw_setup or sm_pi_mw_jxsetup which set the name of the initialization file.

The default value for this argument in the distributed software is Jam7, so the resulting initialization file would be JAM7.INI.

Syntax of Initialization Files

Initialization files are arranged as a list of attributes; each attributes takes a value. The entries take the following form:

attribute=value

example

StackedWindowType=Dialog

The attribute is StackedWindowType. The value, set on the right of the equal sign, is Dialog.

The file is made up of sections; bracketed names indicate each section. Section and content order is not relevant.

Section	Description
[Jam Options]	Behavior and appearance options.
[Jam PaletteColors]	List of names and values for setting the sixteen JAM palette colors.
[Jam NoPaletteColors	Names and Values of colors if display device does not have color palette.

Section	Description
[Jam StatusLine]	Configures how the status line appears.
[Jam Help]	Specifies behavior of JAM interface to Windows Help.
[Jam DDE]	Supports end-user DDE client links.

Comments are indicated with a semicolon at the start of the line.

Colors

JAM provides sixteen basic colors — eight highlighted and eight unhighlighted. You can map these colors to any of the colors supported by the GUI. The mapping between JAM colors and GUI colors defines your color palette.

To create custom colors beyond the provided sixteen basic colors, use the configuration map file (refer to Chapter 8). (Since your users have access to configuration maps, you could allow them to customize the colors.)

Most monitors support sixteen primary colors, but some support more. These sixteen primary colors are mapped to the palette colors in the jam.ini file, which is the initialization file distributed with JAM. The sixteen basic colors are:

black	red	hi_black	hi_red
blue	magenta	hi_blue	hi_magenta
green	yellow	hi_green	hi_yellow
cyan	white	hi_cyan	hi_white

Color Section Syntax If your application is running on a display device which has the Windows color palette enabled, JAM colors are mapped to your Window colors in the [Jam PaletteColors] section of the initialization file. If your application is running on a display device which does not support a color palette, or you have disabled the color palette using the PaletteUse option in the JAM Options section, JAM colors are mapped in the [JAM NoPaletteColors] section. The syntax for both color sections is as follows:

jamcolor = color

Note: There is a limitation in Windows for colors used as foregrounds. Foreground colors must be primary colors (i.e., no dithered patterns). If you specify a non-primary color, Windows rounds it up to a primary color.

Chapter 9 Setting Windows Defaults

jamcolor

A basic JAM color as listed above.

color

An RGB value in the form *red/green/blue* where *red, green* and *blue* are numbers between 0 and 255. For example, Blue=0/0/255 You can use the Windows palette feature on the Windows Control Panel to interactively mix your colors, and then note the values and transfer them to the initialization file.

Note: In Windows JAM screens and widgets having highlighted background colors are different from unhighlighted background colors. In character JAM on a PC under DOS, there is normally no difference between highlighted and unhighlighted background colors.

Initialization Options

The following behavior and appearance options can be set in the [Jam Options] section of the application-specific initialization file.

3D = Yes | No

When this option is set to Yes, the 3D feature is enabled for the application. Message boxes and Windows common dialog boxes, as well as application screens, take on a three-dimensional appearance. When this option is set to No, the 3D feature is disabled for the application. The 3D option setting in this file can be overridden for individual application screens through the 3D screen property; message boxes and Windows common dialog boxes, however, will always take their appearance from the setting in this initialization file. (Refer to Chapter 14 in the *Editors Guide* for an explanation of the 3D property.) If this option is not specified, it defaults to Yes.

FrameTitle = TitleString

This setting controls the title text in the MDI frame around a JAM application. The default title string is the value of the first argument to sm_pi_mw_jxinit or sm_pi_mw_init in jmain.c or jxmain.c.

HideHiddenWindows = Yes | No When set to Yes, this option hides screen editor windows when it enters Test mode.

IconPosition = Xposition Yposition

This options sets the location of the application's minimized icon. *Xposition* and *Yposition* are set in pixels.

IntroPixmap = *image-file*

Specifies the image that appears during application startup. Refer to page 266 in the *Editors Guide* for supported image file types. The specified image file name can include an explicit path; refer to page 267 in the *Editors Guide* for a description of the search algorithm JAM uses when you omit a path.

KeepInMDI = Yes | No

If set to Yes, this option forces initial placement of a newly created window within the MDI frame, if its size allows. No, the default, allows initial placement of the window anywhere within the MDI frame.

MDIWallpaperPixmap = image-file
MDIWallpaperStyle = Center | Tile

These options let you define a wallpaper image and its position on JAM's MDI parent window. Refer to page 266 in the *Editors Guide* for supported image file types. The specified image file name can include an explicit path; refer to page 267 in the *Editors Guide* for a description of the search algorithm JAM uses when you omit a path.

When MDIWallpaperStyle is set to Center, the image is centered on your JAM MDI screen. When MDIWallpaperStyle is set to Tile, the image is placed in the upper left corner of the window and is repeated as many times as is necessary to fill the entire window.

MoveThreshold = Distance

The number of pixels which the cursor must be moved in order to be considered a drag instead of a click. This applies when dragging the rubberband, or creating, moving, or resizing an object.

NormalPosition = Xposition Yposition

This options sets the location of the application MDI frame when it is 'restored'. *Xposition* and *Yposition* are set in pixels.

NormalSize = Width Height

This option sets the size of the MDI frame in when it is 'restored'. *Width* and *Height* are set in pixels.

PaletteUse = Enable | Disable

If set to Enable, JAM uses the color palette to produce colors, allowing the full range of RGB colors supported by your display device without having to make use of dithering. If set to Disable, JAM does not make use of the color palette. Your

Chapter 9 Setting Windows Defaults

application is limited to the reserved system colors (20 on a 256-color display) and dithered combinations of them. This setting requires fewer resources. (You should only set this option to Disable if you tend to run other applications that are very color intensive.)

SaveStateOnExit = Yes | No

When this option is set to Yes the size and location of the MDI frame is saved between sessions. The default setting is Yes.

SMTERM = TerminalType

This option overrides the SMTERM environment variable for JAM applications running under Windows. It allows both DOS and Windows to use JAM without the need to change the environment. To take advantage of this feature, set SMTERM to mswin in the initialization file, and to a DOS terminal type in the environment or SMVARS file. Example DOS terminal types are: cga, ega, mono, softcol and softbw.

SMUSER = Username

This JAM environment variable sets the developer's name for use with multi-user libraries. JAM first looks for user identification from your configuration management tool, if any; failing that it looks at SMUSER; if this variable is not defined it looks for LOGNAME, then USER; if it cannot identify you in any of these ways, it prompts for a user name.

SMVARS = Pathname, SMPATH = Pathname and SMBASE = Pathname

Each of these variables can be set in the initialization file in order to override environment variable settings, permitting different settings for different applications running in the same environment. (Refer to Chapter 4 for detailed information on the meaning of these configuration variables.)

StackedWindowType = Dialog | MDI

When set to Dialog, this option forces all stacked windows in a JAM application to be dialog boxes. As dialog boxes, they are modal, have a dialog style border, and can move outside of the MDI frame. This might lead to unpredictable behavior if the application moves stacked and sibling windows on the window stack, or if the application creates or breaks sibling relationships between windows after they are displayed.

StartupState = Minimized | Normal | Maximized This option controls the size of the MDI frame when JAM starts. Windows will determine the initial size if no value is given.

Status Line Appearance

The [JAM StatusLine] section configures how the status line appears. By default, the status line is an etched out bar with the same colors that Windows uses for push buttons.

Flags = STATUSBARCOLORS PURECOLORS ETCHEDIN

This option sets flags for the status line. STATUSBARCOLORS enables the four status bar color options defined below. If this flag is not set, the control panel color settings for buttons are used. PURECOLORS forces only non-dithered colors to be used in painting the status bar. ETCHEDIN causes the status bar to appear etched into the MDI frame, rather than the default of appearing etched out.

ShadowWidth = Width

This option sets the width of the 3-D status bar shading in pixels. It defaults to 2. If set to 0, there is no 3-D effect.

BackColor = Color TextColor = Color HighLightColor = Color ShadowColor = Color

These options only take effect if the Flags option is set to STATUSBARCOLORS They set the default background color, text color, shaded highlights, and darkened shadow for the status line. For compatibility with other Windows applications, background color defaults to grey. The other colors' default values are calculated by sm_ManipulateColor. Messages with embedded display attributes can override the default background color. *Color* is an RGB value in the form *red/green/blue* where *red*, *green* and *blue* are numbers between 0 and 255. For example, Blue=0/0/255 You can use the Windows palette feature on the Windows Control Panel to interactively mix your colors, and then note the values and transfer them to the initialization file.

Help Behavior

The [JAM Help] section specifies the behavior of the JAM interface to the Windows help engine.

```
WinHelp = On | Off
```

When set to On the F1 key will be trapped by JAM and used to access the help database specified in the HelpFile option, below. Shift-F1 will cause the mouse

Chapter 9 Setting Windows Defaults

cursor to change indication help mode, and the subsequent mouse selection will cause the help database specified in the HelpFile option to be accessed for help. This means that F1 and Shift–F1 will not be available as keystroke to JAM itself. This option defaults to Off.

HelpFile = Pathname

This option specifies a path to the winhelp database file to use for help. It can be a fully specified path, or it can be a file that is somewhere on the path specified by the SMPATH environment variable. There is no default. Therefore this option must be specified if the WinHelp option is set to On.

DDE

This section supports DDE client and server links. For general information on setting DDE client/server links between JAM and other applications, refer to Chapter 31 in the *Application Development Guide*

DDEServer = On | Off When set to On, this option enables JAM as a server.

DDEClient = On | Off When set to On, this option enables JAM as a client.

screenname ! fieldname = service | topic ! item Specifies hot links to server applications. The format for server, topic, and item arguments is specific to the server application. For example, a link to a Quattro Pro spreadsheet might look like this:

salesScrn!totalSales=QPW|C:\myAcct\sales.wb1!\$A:\$A\$10..\$A\$10

Table 58 shows the syntax used by three widely used Windows applications. Refer to the server application's documentation for information on server argument formats. For more information on setting JAM as a DDE client, refer to page 548 in the *Application Development Guide*.

Windows Control Panel

Default attributes for Windows can be set from the Windows Control Panel, usually found in the Main group on the Windows desktop. From the Control Panel,

you can set up the color scheme for Windows, as well as other defaults. The Control Panel alters the win.ini file, supplied by Microsoft. Refer to the MS Windows documentation for details of how to use the control panel

Sample JAM.INI File

[JAM Options]

```
; The following are various general PI/Windows options.
; Each option is preceded with a comment that indicates
; what it is for.
; Set this option to the name you want in the MDI Frame
; Window's title bar. Set this to an appropriate value for
; your application.
FrameTitle=JAM for Windows
; Name of graphic file to display on application startup
IntroPixmap=
; Set SMBASE, SMVARS, SMPATH and SMUSER here to override the
; environment variable settings. This permits different
; settings for different applications running in the same
; environment, without needing to restart Windows.
; Setting any of these options to an empty value will
; maintain the environment settings.
SMBASE=
SMVARS=
SMPATH=$SMBASE\config;$SMBASE\samples\videobiz
SMUSER=
; Use this option to override the SMTERM environment variable
; under Windows. This allows the use of JAM in both DOS and
; Windows without changing the environment.
SMTERM=mswin
; Set this option to "No" to remove the 3-D effect for
; all user screens and system dialogs. Defaults to "Yes"
; if unspecified.
3D=Yes
; Options that control the size and location of the MDI frame
; when JAM starts. If no values are given, initial size &
; location are determined by Windows.
; Set this option to "Minimized", "Normal" or "Maximized".
; Defaults to "Normal".
```

Chapter 9 Setting Windows Defaults

StartupState=Normal

```
; This option sets the normal (i.e. "Restored") size
; (width height) of the MDI frame in pixels. For example,
; NormalSize=100 150 makes the MDI frame 100 pixels
; wide and 150 pixels high when in a restored
stateNormalSize=858 686
; This option sets the normal (i.e. "Restored") location
; (x y) of the MDI frame in pixels.
NormalPosition=24 37
; This option sets the location (x y) of the application's
; minimized icon in pixels. Use this to override Windows'
; default placement.
;IconPosition=
; Set this option to "Yes" to save the size and location of
; the MDI frame in this file each time you exit JAM. Any
; uncommented settings for StartupState, NormalSize,
; NormalPosition and IconPosition are overridden
; upon exiting. The default is "No" if unspecified.
SaveStateOnExit=Yes
; Name of the graphic file to display as the MDI frame
; background. Style determines if graphic is "Tiled" or
; "Centered".
MDIWallpaperPixmap=
MDIWallpaperStyle=
; The next option controls how JAM makes use of the color
; palette, if your display device supports one. Use of the
; palette allows the full range of RGB colors supported by
; your display device to be displayed without dithering. If
; several applications which use the color palette are
; running, they share the palette. The application which is
; active is guaranteed to display its colors correctly. If
; applications share the palette correctly, applications
; which are not active get colors which are as close as
; possible to the requested colors. More information on color
; palettes can be found in chapter 19 of the Windows 3.1 SDK
; "Guide to Programming".
; If set to Disable, JAM does not make use of the color
; palette. Your application is limited to the reserved system
; colors (20 on a 256 color display) and dithered
; combinations of them. This setting requires fewer
; resources. In this configuration, the colors displayed by
```

; JAM are independent of the focus. You should only set this ; option to Disable if you tend to run other applications

JAM 7.0 Configuration Guide

```
; that are very color intensive.
; Set this option to "Enable" or "Disable". Default is
; "Enable".
PaletteUse=Enable
; The following option can be used to force all stacked
; windows in a JAM application to be dialog boxes. This
; makes them modal, gives them a dialog style border, and
; allows them to move outside of the MDI frame. Note that
; this may lead to unpredictable behavior if the application
; moves stacked and sibling windows around on the window
; stack, or if the application "siblingizes" and/or
; "desiblingizes" windows after they are displayed.
; Set this option to "Dialog" to force stacked windows to be
; dialog boxes or to "MDI" if you want to use regular MDI
; windows. "Dialog" is the default. Note that in JAM 5 the
; default value was "MDI".
StackedWindowType=Dialog
; Setting this option to "Yes" (default) causes Edit Mode
; windows to be hidden when going onto test mode. "No" leaves
; them visible.
HideHiddenWindows=Yes
; The option "MoveThreshold" is the number of pixels which
; the mouse must be moved in order to be considered a drag
; instead of a click. This applies when dragging out a
; rectangle to create an object, or when trying to
; move or resize an object. If you actually do want to move
; or resize an object by less than this amount, drag a larger
; distance first, then drag back to the desired position.
; (Do not release the mouse in between.) This only takes
; effect in Edit Mode.
MoveThreshold=5
; Setting this option to "Yes" causes new MDI windows to be
; positioned within the MDI frame, if possible. "No"
; (the default) places no constraints on the initial position
; of windows.
KeepInMDI=No
; The following two options disable JAM as a DDE client
; or server. By default, JAM is enabled as both a client
; and server. Set these to "Off" to disable either or both.
DdeClient=On
DdeServer=On
[JAM PaletteColors]
```

Chapter 9 Setting Windows Defaults

```
; The colors in this section are used if the display device
; supports a color palette, and you have not disabled use
; of the color palette using the PaletteUse option. See the
; comments prior to PaletteUse in the [Jam Options] section
; for more information.
; This section matches the 16 JAM basic colors to RGB values.
; It is not optional, as JAM must have a way to perform this
; mapping. The 16 colors listed here correspond to the
; colors of the screen editor's Color Palette screen. You can
; change any of these values to override the default.
; Values for these colors are decimal RGB values. You can
; run Control Panel - Colors to look at colors by decimal
; RGB value.
black=128/128/128
blue=0/0/191
green=0/191/0
cyan=0/191/191
red=191/0/0
magenta=191/0/191
yellow=191/191/0
white=191/191/191
hi_black=0/0/0
hi_blue=0/0/255
hi_green=0/255/0
hi_cyan=0/255/255
hi_red=255/0/0
hi_magenta=255/0/255
hi_yellow=255/255/0
hi_white=255/255/255
[JAM NoPaletteColors]
; Colors in this section are used if the display device does
; not support a color palette, or you disabled use of the
; color palette using the PaletteUse option. See comments
; prior to PaletteUse in the [Jam Options] section for more
; information.
; This section matches the 16 JAM Basic colors to RGB values.
; It is not optional: JAM must perform this mapping. The
; 16 colors listed here correspond to the colors of the
; screen editor's Color Palette screen. You can change any of
; these values to override the default. Values for these
; colors are decimal RGB values. You may run Control
; Panel - Colors to look at colors by decimal RGB value.
black=128/128/128
blue=0/0/128
green=0/128/0
```

```
cyan=0/128/128
red=128/0/0
magenta=128/0/128
yellow=128/128/0
white=192/192/192
hi_black=0/0/0
hi_blue=0/0/255
hi_green=0/255/0
hi_cyan=0/255/255
hi_red=255/0/0
hi_magenta=255/0/255
hi_yellow=255/255/0
hi_white=255/255/255
[JAM StatusLine]
; This section configures how the status line appears. By
; default, the status line is an etched out bar with the same
; colors that Windows uses for push buttons. The following
; options modify its appearance:
; The "Flags" option sets flags for the status line.
; Available flags are:
; STATUSBARCOLORS - If this flag is set, the options
;
                     BackColor, TextColor, HiLightColor,
                     and ShadowColor below are meaningful.
;
;
                     If it is not set, colors used for the
                     status bar are those set for buttons in
;
                     the control panel.
;
;
 PURECOLORS
                   - all colors used for painting the status
                     bar are forced to pure non-dithered
;
                     colors.
;
 ETCHEDIN
                   - The status bar is etched in to the MDI
;
                     frame, rather than being etched out.
;
                     Etched out is the default.
;
;
; By default, no flags are set.
Flags=
; The "ShadowWidth" option sets the width of the 3D status
; bar shading in pixels. It defaults to 2. If set to 0,
; there is no 3D effect.
ShadowWidth=2
; The "BackColor", "HighLightColor", and "ShadowColor"
; options are the colors used to paint the status bar
; background, shaded highlights, and darkened shadow. These
; options only take effect if STATUSBARCOLORS is
; one of the flags specified in the "Flags" option above.
; The colors can be GUI independent colors. "BackColor"
```

Chapter 9 Setting Windows Defaults

```
; defaults to gray. "TextColor", "HighlightColor" and
; "ShadowColor" are calculated using sm_ManipulateColor()
; if they are not specified here.
BackColor=128/128/128
;TextColor=0/0/0
;HighLightColor=255/255/255
;ShadowColor=0/0/0
; The "Font" option sets a font for use on the status line.
; It can be one of the system fonts, like SYSTEM_FONT, or
; ANSI_VAR_FONT, or it can be a font description in the form:
; fontname-size[-bold][-italic][-underline]
; or it can be a GUI-independent font. It defaults to
; SYSTEM_FIXED_FONT.
Font=MS Sans Serif-10
[JAM Help]
; This section specifies the behavior of the JAM interface to
; external help engines.
; This option contains the name of the help file used by the
; external help system you've installed.
HelpFile=
; The option "EditorHelpPath" specifies a path where the
; directory that contains the book directory that contains
; the editors directory can be found. This should be removed
; when providing an .ini file with your finished application.
EditorHelpPath=$SMBASE\docs\books
[JAM DDE]
; This section supports end user DDE client links. Specify
; the JAM screen name, JAM widget name, the remote server
; name, the topic, and the item.
; Example:
; scrnname.jam!widgetname=Excel|Sheet1!R1C1
; The screen name MUST be separated from the widget name with
; an '!'. The Server, Topic and Item MUST be separated with
; '|' and '!' characters.
;WARNING:QuattroPro requires full path name for its topic.
         For example, to connect linkfunc.jam client topic
;
;
         and DDETextField client item with QPW server
         FROMINI.WB1 server topic, and A1 server item,
;
;
         write:
;
```

linkfunc.jam!DDETextField=QPW|C:\QPW\FROMINI.WB1!A1 ; ;NOTE: JAM topic and item can be specified in lower, upper, or mixed cases. QuattroPro and Microsoft Excel servers ; allow specification of names of their servers, topics, ; and items in lower, upper, or mixed cases. ; [JAM EditMenu] ; This section lets users set Edit Menu labels, mnemonics, ; and accelerators. Below are examples of default settings ; which are commented out. ;WARNING: in case a user changes default settings for ACCELERATORS, he/she should change corresponding ; items in mwjxform.rc file. ; ;CutLabelAndMnemonic=Cu&t ;CutAccelerator=CtrlDel+X ;CopyLabelAndMnemonic=&Copy ;CopyAccelerator=Ctrl+C ;PasteLabelAndMnemonic=&Paste ;PasteAccelerator=Ctrl+V ;DeleteLabelAndMnemonic=Delete ;DeleteAccelerator=Del ;ClearLabelAndMnemonic=Clear ;ClearAccelerator= ;SelectAllLabelAndMnemonic=&Select All ;SelectAllAccelerator= [JAM WindowsMenu] ;This section lets a user set Windows Menu labels, mnemonics, ; and accelerators. Below are examples of default settings ; which are commented out. WARNING: In case a user changes default settings for ; ACCELERATORS, he/she should add corresponding ; items in accelerator table in mwjxform.rc file. ;CascadeLabelAndMnemonic=&Cascade ;CascadeAccelerator= ;TileLabelAndMnemonic=&Tile ;TileAccelerator= ;ArrangeIconsLabelAndMnemonic=Arrange &Icons

;ArrangeIconsAccelerator=

Chapter 9 Setting Windows Defaults


Setting Motif Defaults

Applications that run under Motif use resource files to set defaults for the GUI (Graphical User Interface). The resource file controls how Motif and the application running under Motif appear and behave. You can set up the initial state, and as is the practice in Motif, your users can change these settings to suit their preferences.

Resource Files

Resource preferences are indicated by setting attribute/value pairs. JAM applications use resource files to determine values for a variety of attributes including:

- Default colors.
- Mapping between JAM colors and Motif colors.
- GUI-independent color names.
- Application behavior.

Resource Filenames

Each application can have an application-specific resource file. The name of this file is determined by the class name for the application. The class name for a JAM

application is set by the argument to JAM's GUI initialization function. To change the class name, edit the argument to the function sm_pi_xm_setup in the file piinit.c for a JAM application executable, or the function sm_pi_xm_jxsetup in the file pijxinit.c for a JAM development executable. These source files can be found in the link directory.

At initialization, the main routine of your application (usually either jmain.c or jxmain.c) calls either the function sm_pi_init or sm_pi_jxinit to initialize the GUI. These routines in turn call sm_pi_xm_setup or sm_pi_xm_jxsetup which set the class name.

The default class name in the distributed software is XJam. Therefore the application-specific resource filename is XJam.

Structure of Resource Files

Under Motif, resource files are arranged as colon separated attribute/value pairs. For example:

XJam*stackedWindowsAreDialogs: false

The attribute set in this case is stackedWindowsAreDialogs. The value is understood to be any text to the right of the colon. White space directly after the colon is ignored. Therefore the value is false.

XJam is the class name. It restricts this resource to applications of the class XJam. Resources may be further restricted to screens and even to individual widgets. The class name for a JAM application is determined at application initialization (see above). You can also specify an instance name for an application via the standard Xt command line switch -name.

Comments are indicated by starting the line with an exclamation point. Refer to your Motif documentation for a full explanation of resources and resource files.

Location of Resource Files

A resource database is constructed from several sources, in the following order of precedence, each source overriding any conflicting settings from the previous sources:

The application-specific resource file, named by the class name of the application, is sought in the directory: /usr/lib/X11/app-defaults on the client machine. These resources are then global to all users of a particular application.

- An application-specific resource file is then sought in the user's home directory. The user may override the global setting here.
- If the environment variable XAPPLRESDIR is set, the directory named in it on the client machine is searched for a resource file named by the application class name. This file can contain the user's or site administrator's preferences, and overrides settings in both the application-specific resource file in the app-defaults directory and the user's home directory.
- Resources that are particular to one user's preference can be included in the .Xdefaults file in the user's home directory. The .Xdefaults takes precedence over other resource files. If you make changes to the .Xdefaults file while the Motif Window Manager is running, you must call xrdb -load .Xdefaults to reload the resource file.
- Command line options override any resources set in a resource file.

Colors

JAM running under a GUI offers access to many more color choices than character JAM. Resource files provide a mapping between JAM colors and Motif colors. JAM also provides a way to set up a GUI-independent color naming scheme in the resource file. These colors can be used in widget and screen properties.

Setting Palette Colors

Character JAM provides sixteen colors from which you can choose, eight highlighted and eight unhighlighted. In the resource file, you can map these sixteen JAM colors to any of the colors supported by Motif. This mapping between JAM colors and Motif colors define your palette. Since your users have access to resource files, they can customize the palette. The sixteen JAM colors that can be defined in the palette are:

black	red	hi_black	hi_red
blue	magenta	hi_blue	hi_magenta
green	yellow	hi_green	hi_yellow
cyan	white	hi_cyan	hi_white

In JAM running under Motif, palette colors are mapped to Motif colors defined in the resource file. The syntax is:

Chapter 10 Setting Motif Defaults

XJam.jamcolor: color

jamcolor

A basic JAM color as listed above.

color

Can be either

• A color name that appears in the rgb.txt file on your system. For example:

XJam.blue: DarkSlateBlue

• A hexadecimal RGB value. Hex specifications must be preceded by a # symbol. For example:

XJam.green: #00a800

Refer to the Motif User's Guide for details.

Colors Beyond the JAM Palette

For most applications, sixteen colors are sufficient. However, if additional colors beyond the sixteen are needed, you can specify them in the widget or screen properties with the extended color type. With the extended color type, you can use either GUI-specific colors or GUI-independent color aliases.

Overriding Colors Set within JAM

Color Resources Motif provides resources for changing the color of widgets and these setting can override any color settings made within JAM. For example, a foreground color setting for a text widget might look like this:

XJam*XmText*foreground: blue

This setting overrides any other foreground color for text widgets in applications of class XJam. A setting like the following changes the text widget for the specified screen vidscreen:

XJam*vidscreen*XmText*foreground: blue

Background and Foreground Resources Motif provides application-wide background and foreground color resources. You can set these from the command line or in the resource file. JAM interprets these resources to override JAM's default background and foreground colors. Therefore, the application-wide background color replaces any un-highlighted black background, and the application-wide foreground color replaces any unhighlighted white foreground.

JAM 7.0 Configuration Guide

The format in the resource file is:

XJam*background: *color* XJam*foreground: *color*

To set the colors from the command line, the format is:

-bg color -fg color

color

Can be either

- Motif color name
- Hex value preceded by a # symbol.

Note: GUI-independent color aliases cannot be used with these resources.

Background and foreground resources offer a convenient method for allowing users to set their own color preferences, provided that you specify unhighlighted black as the background and unhighlighted white as the foreground in the display attributes for widgets and screens, and that the widgets and screens don't have background or foreground colors specified.

Motif Colors

Motif colors are listed in the rgb.txt file, often found in the directory / usr/lib/X11. If your rgb.txt file is located in a different directory, use the XJam.rgbFilename: *directory* resource to point to it. The rgb.txt file lists color names along with their red, green, and blue components. The colors are system dependent. Some common color names are listed in Table 24:

Table 24. Motif colors (partial listing)

alice blue	deep sky blue	light sky blue	papaya whip
antique white	dim gray	light slate blue	peach puff
aquamarine	dim grey	light slate gray	peru
azure	dodger blue	light slate grey	pink
beige	firebrick	light steel blue	plum
bisque	floral white	light yellow	powder blue
black	forest green	lime green	purple
blanched almond	gainsboro	linen	red
blue	ghost white	magenta	rosy brown
blue violet	gold	maroon	royal blue
brown	goldenrod	medium blue	saddle brown

Chapter 10 Setting Motif Defaults

burlywood	gray	medium orchid	salmon
cadet blue	green	medium purple	sandy brown
chartreuse	green yellow	medium sea green	sea green
chocolate	grey	medium slate blue	sienna
coral	honeydew	medium turquoise	sky blue
cornflower blue	hot pink	medium violet red	slate blue
cornsilk	indian red	midnight blue	slate gray
cyan	ivory	mint cream	slate grey
dark goldenrod	khaki	misty rose	snow
dark green	lavender	moccasin	spring green
dark khaki	lavender blush	navajo white	steel blue
dark olive green	lawn green	navy	tan
dark orange	lemon chiffon	navy blue	thistle
dark orchid	light blue	old lace	tomato
dark salmon	light coral	olive drab	turquoise
dark sea green	light cyan	orange	violet
dark slate blue	light goldenrod	orange red	violet red
dark slate gray	light gray	orchid	wheat
dark slate grey	light grey	pale goldenrod	white
dark turquoise	light pink	pale green	white smoke
dark violet	light salmon	pale turquoise	yellow
deep pink	light sea green	pale violet red	yellow green

Resource Options

This section describes resources that control behavior and the appearance of JAM running under Motif.

Behavioral Resources

Carranal		+ 1	1: - ·			1 1		CTANA.
Several	resources	control	applica	411ON-V	viae.	nenav	10r 0	I IAW'
Deverui	resources	control	appire	across .	1140	o o na i	101 0	

Splash Screen Resource	The introPixmap resource specifies the image that appears during application startup. Refer to page 266 in the <i>Editors Guide</i> for supported image file types. The specified image file name can include an explicit path; refer to page 267 in the <i>Editors Guide</i> for a description of the search algorithm JAM uses when you omit a path.
Base Window Resource	The baseWindow resource controls whether a base window appears on the display. The base window is a special window that contains only a menu bar, a keyset, and a status line.

174

	If bas	seWindow is:	
	O t	crue (default) — a base window appears on the display.	
	O f s a b	Ealse — No base window appears on the display. Any menu bar, keyset, or status line that would appear in this window are lost. Refer to formStatus and formMenus to determine which status line and menu bars appear in the base window.	
Screen Editor Cursor Behavior Resources	The move applie to mo amou back This r	moveThreshold resource sets the number of pixels which the cursor must be ed in order to consider the movement to be a drag instead of a click. This es when dragging out a rubber band when creating an object, or when trying ove or resize an object. If you want to move or resize a field by less than this int, drag the mouse a larger distance than this resource is set to, then drag it to the desired position without releasing the mouse button during the action. resource defaults to 0.	
	The p befor pressi resou mous	eressAndMove resource determines whether an object must be selected to it can be moved. When this resource is set to true (the default behavior), ing on a widget selects it and allows it to be immediately dragged. When this arce is set to false, the widget must first be clicked on (press and release the se), and then pressed on again in order to move it.	
Status Message Resource	The f messa the st windo line. 7 (see a	FormStatus resource controls where status messages appear (not error ages). Error messages appear in dialog boxes, while status messages appear on tatus line. This resource controls whether status messages appear on the base ow's status line (the default), or on the active screen's (or window's) status The existence of the base window is controlled by the baseWindow resource above).	
	There	e are five levels of status messages:	
	1. d	d_msg_line	
	2. v	wait	
	3. f	field	
	4. r	ready	
	5. t	background	
	Background status messages can only appear in the base window. If formstatus is:		
	O f s k	Ealse (default) — All status messages appear in the base window. Individual screens have no status line of their own. If there is no base window (that is, if baseWindow: false), then there is no status line at all.	
Chapter 10 Setting Matif	Defaulte	175	

Chapter 10 Setting Motif Defaults

	• true — Background status messages appear in the base window. All other status messages appear in a status line on the active screen. The status line on individual screens appears at the bottom of the screen. Only the active screen's status line is updated. If a screen is not active, then its status line is not updated.				
Menu Resources	The following resources control menu display:				
	formMenus				
	Controls whether individual screens or windows have their own menu bars. formMenus can be set to true or false:				
	• false (default) — Only the base window displays a menu bar. Individual screens display no menu bar. Menu bars of all scopes, including screen-level, appear in the base window. If baseWindow is also false, then no menu bars appear at all.				
	• true — Individual screens display their own menu bar. Screens display menu bars of the scope MNS_SCREEN (screen-level). Only the active screen's menu bar is updated and active. Menu bars on inactive screens are inactive.				
	The base window, if there is one, displays menu bars of the scope MNS_AP- PLIC (application-level). The SFTS logical key can toggle between having the application-level or system-level menu bar displayed in the base window. If there is no base window, then no system- or application-level menu bars are displayed.				
	Sets the label for the specified edit item—for example, edit_cut or edit_paste. This statement sets the label for edit_cut to Cut:				
	XJam*XmMenuShell*edit_cut.labelString: Cut				
	<pre>mnemonic Sets the mnemonic for keyboard access to the specified edit item—for example, edit_cut or edit_paste. This statement sets for edit_cut's mnemonic to t:</pre>				
	XJam*XmMenuShell*edit_cut.mnemonic: t				
Toolbar Resources	You can control tooltips font type and size for Motif applications through the XJam resource file. For example, this statement sets tooltip text to 18 point Helvetica:				
	XJam*toolbar*tooltip.fontList: *-helvetica-*-18-*				
Combination of Settings	There are combinations of setting you can use with the behavioral resources that will ensure compatibility with Windows and ensure compliance with Motif				

176

standards, with respect to where and how menu bars and status lines appear and work.

For compatibility with JAM running under Windows and backward compatibility with controlled release versions of JAM running under Motif, use the following default settings:

XJam*baseWindow:	true
XJam*formStatus:	false
XJam*formMenus:	false

• For full functionality with menu bars and status lines local to screens:

XJam*baseWindow: true XJam*formStatus: true XJam*formMenus: true

• If you wish to have no base window:

XJam*baseWindow: false XJam*formStatus: true XJam*formMenus: true

Do *not* use application level menu bars or background status messages with this combination; they will not appear.

Screen Control Resource

The focusAutoRaise resource brings a screen to the top of the display when it gets the focus. Use the following setting:

XJam*focusAutoRaise: true

Restricted Resources

The following items in the distributed XJam file *must not* be changed:

XJam*...*translations XJam*keyboardFocusPolicy XJam*...*traversalOn

You can change all other items (including: Mwm*XJam*keyboardFocusPolicy).

Global Resource and Command Line Options

The resources in Table 25 are global settings that function on an application-wide basis. You can also specify them on the command line, as you can the standard X

Chapter 10 Setting Motif Defaults

Toolkit command line options. Refer to the *X Toolkit* manual for a full list of command line switches.

Resource	Туре	Command Line	Description
foreground	string	-bg color	Sets unhighlighted white foregrounds to color.
background	string	-fg color	Sets unhighlighted black backgrounds to col- or.
ownColormap	boolean	-cmap (on) +cmap (off) D	Tells JAM whether to use its own color map. Use 'on' for systems with limited colors.

Table 25. Global resource options

D = default

command line example The following illustrates a sample command line resource setting:

jamdev -fg 'white' myscreen.jam

Widget Hierarchy

Widgets are arranged in a parent-child hierarchy. The tables in this section describe
the widget hierarchy in JAM running under Motif. If you want to set resources for
particular widgets or classes of widgets in your application, you will need this
information. Refer to the OSF/Motif Programmer's Guide for more information on
widgets, widget classes, and the resources associated with them.Base ScreenThe base screen in a JAM application is an ApplicationShell widget. Its class
is given by the first argument to the initialization routine, and its name is the name
of the application program (the value of argv[0] in main). If the baseWindow
resource is set to false, then this shell is created but never displayed.Note:Avoid application program names that contain periods or asterisks, as the
resource parser interprets these as special characters. Periods that precede widget
name extensions are exempt from this restriction: extensions are stripped from JAM
names before Motif uses them.

By default, JAM has class name XJam and application name jamdev or jam.

Table 26 lists the widget hierarchy for the base screen.

Table 26. Base screen widget hierarchy

Name	Widget Class
application-name	ApplicationShell (given by initialization routine)
main	XmMainWindow
statusForm	XmForm
statusText	XmText
menubar	XmRowColumn
toolbar	XmRowColumn

The status area is used for the JAM status line in the base screen.

Dialog Boxes The creation of dialog boxes is handled by JAM, in some cases, and by Motif in others. Table 27 lists the appropriate function or Motif resource associated with the creation of specific dialog box types.

Table 27. Dialog boxes

Dialog box type	Created by
File selection	sm_filebox library function
Message	Need to post message
Error message	XmCreateErrorDialog
Query message	XmCreateQuestionDialog

JAM specifies the message string, which buttons appear, and which button is the default. The JAM message call can specify the icon to appear. Other options, like the title bar text, can be set in the resource file.

The children of dialog boxes are handled by Motif. Refer to your Motif manual for details.

JAM Screens The widgets used in JAM screens are all subclasses of the Motif shell widget. The shell's parent is the ApplicationShell. Table 28 lists the widget hierarchy for JAM screens.

Chapter 10 Setting Motif Defaults

ne 20.	mager merarchy for shin screens					
	Name	Widget Class				
	screen-name	TopLevelShell				

Table 28.Widget hierarchy for JAM screens

screen-name	TopLevelShell	
message_popup	XmDialogShell	
message	XmMessageBox	
filebox_popup	XmDialogShell	
fileBox	XmFileSelectionBox	
scroll	XmMainWindow	
clip	SmSimpleManager	
area	SmSimpleManager	
statusForm	XmForm	
statusText	XmText	
scrollbar	XmScrollBar	
scrollbar	XmScrollBar	
menubar	XmRowColumn	
toolbar	XmRowColumn	

JAM screens have a status line only if the value of the formStatus resource is true. They have a menu bar only if formMenus is true.

New screens are named shell before they have been saved.

Since the name of the shell used for JAM screens is the screen name, resources can be restricted to a specific screen if you begin the specification with *class*screen_name*. For example, XJam*vidscrn... begins a specification for a screen named vidscrn in an application of class XJam. Resources restricted to a named screen are equivalent to screen property. For example:

XJam*vidscrn.background: gold

is the same as specifying a screen color property.

area is the parent widget for all the widgets on a JAM screen. To place your own widgets on a JAM screen, you need the widget ID of area. The library function sm_drawingarea returns the widget ID of area. A related function, sm_translatecoords, translates JAM screen coordinates into pixel coordinates relative to the upper left hand corner of area.

Widgets JAM widgets are created as child widgets of area. If a widget has a name, its corresponding Motif widget gets the same name. If a field doesn't have a name, its

JAM 7.0 Configuration Guide

Motif widget is named _fld#, where # is the field number. In a named array consisting of multiple fields, each widget has the same name. Motif widgets that represent multiple fields take the name of the first field.

Motif variants of sm_widget such as sm_xm_widget return a widget ID. Asterisks in the table below indicate which widget is returned by sm_widget in cases where there is more than one possibility. If the widget returned by sm_widget is not the one you are looking for, use XtParent to obtain the widget ID of its parent. This is particularly useful when working with scale widgets and scrolling multiline and list box widgets.

Table 29 lists the widget hierarchy for JAM fields. Some entries in the table have prefixes or suffixes with their names. For example, *field–namesw* indicates that the widget's name is composed of the field's name followed by the characters SW.

Object	Name	Widget Class
box	box	XmFrame
	title	XmLabel
line	separator	XmSeparator
graph	box	XmDrawingArea
grid	grid-name	SmMatrix
	horizScroll	XmScrollBar
	vertScroll	XmScrollBar
	clip	SmClip
	textField	XmText
single line text	field–name	XmText
static label	field–name	XmLabel
radio button	field–name	XmToggleButton
toggle button	field–name	XmToggleButton
check box	field–name	XmToggleButton
dynamic label	field–name	XmLabel
push button	field–name	XmPushButton

Table 29. Widget hierarchy for JAM widgets

Chapter 10 Setting Motif Defaults

Object	Name	Widget Class
combo box	field–name	XmForm
	text	XmText
	arrow	XmArrowButton
	field-name_pane	XmRowColumn
	label-text label-text label-text	XmPushButton XmPushButton XmPushButton
multiline text	field–name	XmText
multiline text with scrollbars	field–namesw	XmScrolledText
	field–name	XmText*
list box	field–name	XmList
list box with scroll bars	field-namesw	XmScrolledList
	field–name	XmList*
option menu	field–name	XmRowColumn*
	popup_ <i>field_name</i> _pane	XmMenuShell
	field-name_pane	XmRowColumn
	label-text label-text	XmPushButton XmPushButton
	label-text	XmPushButton
scale	field–name	XmScale
	scale_scrollbar	XmScrollBar*

To refer to a whole class of widgets, use the widget class. For example, XJam*XmText refers to all text widgets. To refer to a class of widgets on a screen, use the screen name followed by the widget class. For example, XJam*empscreen*XmText refers only to text widgets on the screen empscreen. To refer to an individual widget, use the screen name followed by the widget's name. For example, XJam*empscreen*empname refers only to the empname widget on the screen empscrn.

In the option menu widget, the text field and the pop-up pane are linked through the subMenuID field of the RowColumn widget. Since the push buttons in the

JAM 7.0 Configuration Guide

option menu are named by their contents, it is easier to set a resource for all the push buttons in an option menu than it is to set a resource for an individual button.

Menus and Toolbars Menus—instantiated as menu bars and their submenus, as pop-up menus, or as toolbars—are created within RowColumn widgets. Menu bars are children of either the base screen's or an individual screen's MainWindow. Submenus are children of MenuShells, but the name of the shell is unclear, since Motif reuses these shells. If a new shell is created, its name is popup_submenu-name. Specify resources for a submenu by using the form: XJam*XmMenuShell.submenu-name. Table 30 lists the hierarchy for menus and pop-up menus.

Object	Name	Widget Class
menu bar	menu–name	XmRowColumn
submenu	(name varies)	XmMenuShell
	submenu–name	XmRowColumn
pop-up menu	application-name	ApplicationShell
	dummy	TransientShell
	popup_popupmenu	XmMenuShell
	popupmenu	XmRowColumn
toolbar	toolbar	XmRowColumn

Table 30. Hierarchy for menus and pop-up menus

Submenus pop up through the auspices of a CascadeButton widget. A submenu is tied to its CascadeButton via the XmNsubMenuID field of the button.

Menu items are children of the menu's RowColumn widget. Table 31 lists their hierarchy.

Menu script keyword	Name	Widget class
ACTION	label-text	XmPushButton
EDCLEAR	edit_clear	XmPushButton
EDCOPY	edit_copy	XmPushButton
EDCUT	edit_cut	XmPushButton

Table 31. Hierarchy for menu item types

Chapter 10 Setting Motif Defaults

Menu script keyword	Name	Widget class
EDDEL	edit_delete	XmPushButton
EDPASTE	edit_paste	XmPushButton
EDSELECT	edit_select	XmPushButton
SUBMENU	label-text	XmCascadeButton
SEPARATOR	separator	XmSeparator
TOGGLE	label-text	XmPushButton
WINLIST	window-name	XmPushButton
	window-name	XmPushButton
	 window-name	XmPushButton
WINOP	windows_raise	XmPushButton

Toolbar items are children of the menu's RowColumn widget. Table 32 lists their hierarchy.

Table 32. Hierarchy for toolbar components

Object	Name	Widget class
button	label-text	XmPushButton
separator	_tool#	XmDrawingArea

Sample Motif Resource File for JAM

JAM 7.0 Configuration Guide

! Set Geometry for the base window. ! Default 600+0+0 XJam.geometry: ! Use this as example if Interactive Placement is set to true !XJam.geometry: 600 !### Double Click Resources ### ! Set preference for double click time in thousandths of a ! second (e.g. 500 = 1/2 second). If double clicks are ! are treated as separate clicks, raise this value. If ! separate clicks are unexpectedly treated as a double click, ! lower this value. XJam.multiClickTime: 400 !### Color Resources ### ! Set the location of the X rgb color database file. ! This must be changed to reflect the site-specific location. XJam.rgbFileName: /usr/lib/X11/rgb.txt ! Set any GUI colors in your color scheme. XJam*foreground: white XJam*background: dark slate gray ! Set the 16 Jam colors, so JAM can map these colors to ! actual GUI values. Modify the values to suit your needs. XJam.black: #000000 XJam.blue: #0000a8 XJam.green: #00a800 XJam.cyan: #00a8a8 XJam.red: #a80000 XJam.magenta: #a800a8 XJam.yellow: #a85400 XJam.white: #a8a8a8 XJam.hi_black: #545454 XJam.hi_blue: #5454ff XJam.hi_green: #54ff54 XJam.hi_cyan: #54ffff XJam.hi_red: #ff5454 XJam.hi_magenta: #ff54ff XJam.hi_yellow: #ffff54

#ffffff

Chapter 10 Setting Motif Defaults

XJam.hi_white:

!### Screen Editor Resources ### ! Resources in this section only affect the operation of ! JAM Screen Editor. Remove when supplying a resource ! file to your end-users. ! Determines how many pixels mouse pointer must move in Edit ! Mode before being considered a drag. XJam.moveThreshold: 6 ! Set the path of the Editor help file. Set in order to ! use online help in the screen editor. XJam.editorHelpPath: \$SMBASE/docs/books ! Splits Property Window drop down into two or more columns ! if more than 15 items appear, to ensure that entire list is ! visible. Users of high-resolution monitors may use a higher ! threshhold as the value of numColumns: 30 on a 17" ! 1024x768 monitor for instance. XJam*smpropty*proptext_pane*packing: PACK_COLUMN HORIZONTAL XJam*smpropty*proptext_pane*orientation: XJam*smpropty*proptext_pane*numColumns: 15 !### General Resources ### ! Point this resource to an XPM, GIF or JPEG file to display ! when the application starts up. XJam.introPixmap: ! Set to false to remove extra insert cursors that Motif 1.2 ! displays by default. Defaults to true if not specified. XJam*cursorPositionVisible: false ! Uncomment the following line to enable Tear-Off Menus. ! Tear-Off Menus are disabled by default. !XJam*tearOffModel: TEAR_OFF_ENABLED ! Set the following resource to true to force all JAM stacked ! windows to be application modal dialogs. Defaults to false.

XJam*stackedWindowsAreDialogs: false

JAM 7.0 Configuration Guide

! Example of specifying the font for the tooltip text.

! XJam*toolbar*tooltip.fontList: fixed

! The following prevents file selection boxes from changing ! size when Filter button is pressed. This may be changed ! according to taste. Default is RESIZE_ANY if not specified.

XJam*XmFileSelectionBox.resizePolicy: RESIZE_NONE

! Set full path and name of runtime help file. This option ! contains name of the help file used by external help system ! you've installed. The default value is empty.

XJam.helpFile:

! Under VMS, text widgets seem to grab the selection ! unless the following is set.

XJam*area*navigationType: NONE

! Drag and drop protocol in many commercial Motifs can be ! buggy . Also, it increases widget creation overhead and may ! affect performance. These resources are set to disable ! Motif drag and drop. This does NOT affect the ability to ! drag and drop widgets within the JAM Screen Editor. ! Enable these resources by changing their values.

XJam*dragReceiverProtocolStyle:	DRAG_NONE
XJam*dragInitiatorProtocolStyle:	DRAG_NONE
XJam*dropSiteActivity:	DROP_SITE_INACTIVE

! Text and mnemonics used in window and edit menu items.

XJam*XmMenuShell*windows_raise.labelString	:	Raise All
XJam*XmMenuShell*windows_raise.mnemonic:		R
XJam*XmMenuShell*edit_cut.labelString:		Cut
XJam*XmMenuShell*edit_cut.mnemonic:		t
XJam*XmMenuShell*edit_copy.labelString:		Сору
XJam*XmMenuShell*edit_copy.mnemonic:		С
XJam*XmMenuShell*edit_paste.labelString:		Paste
XJam*XmMenuShell*edit_paste.mnemonic:		P
XJam*XmMenuShell*edit_delete.labelString:		Delete
XJam*XmMenuShell*edit_delete.mnemonic:		D

Chapter 10 Setting Motif Defaults

```
XJam*XmMenuShell*edit_select.labelString:
                                              Select All
XJam*XmMenuShell*edit_select.mnemonic:
                                              S
XJam*XmMenuShell*edit_clear.labelString:
                                              Clear
XJam*XmMenuShell*edit_clear.mnemonic:
                                              1
! The standard JAM X key file "xwinkeys" maps unmodified,
! shifted, and control function keys 1-12 to JAM logical keys
! PF1-12, SPF1-12, and SFT1-12. This conforms to standard key
! conventions used for JAM on character terminals.
1
! These might conflict with the fallback or vendor-
! specific default bindings Motif uses for virtual keysyms.
! The following line disables all virtual keysyms in a JAM
! application. (The default binding for osfMenuBar is
! remapped to F25. If we unmap it, the Motif library
! resets it to F10.)
1
! If you prefer the standard Motif usage for function keys,
! change the JAM key file to avoid keys that conflict with
! Motif. The following line can then be commented-out.
!
! If you retain any of the following, retain entries for both
! osfMenu and osfMenuBar; otherwise, the program crashes
! on some versions of Motif. (You can change which keys they
! are bound to.)
XJam*defaultVirtualBindings: \n\
   osfMenu:
                              <Key>F26 \n\
   osfMenuBar:
                              <Key>F25 \n\
   osfActivate:
                              <Key>KP_Enter \n\
   osfCancel:
                              <Key>Escape \n\
   osfDown:
                              <Key>Down \n\
   osfLeft:
                              <Key>Left \n\
                             <Key>Right \n\
   osfRight:
                        <Key>Up \n\
<Key>BackSpace \n\
   osfUp:
   osfBackSpace:
   osfDelete:
                             <Key>Delete \n\
```

<Key>F1

JAM 7.0 Configuration Guide

osfHelp:



Index

Symbols

(pound sign)
 in key translation file, 77
 in video file, 92

% (percent sign) in message file, 57 parameter sequences, 94

%A, display attributes in messages, 57

%B, bell for messages, 59

%K, key label in message, 59

%Md, force user acknowledgment of messages, 59

%Mu, acknowledgment of error messages, 60

%N, carriage returns in messages, 60

%W, pop-up window for messages, 60

@date, defining format for, 66

 \setminus (backslash), in messages, 44

Numbers

3D, Windows initialization option, 156

Α

AC_KEEPATTRS, 31 AC_SETATTRS, 31 AC_SWATTRS, 31 ADDM key (add mode), hex value, 80 Aliasing, colors, 138-142 ALT keys, hex value, 81 Alternate character sets, 125-128 ANSI terminal latch attributes, 120 sample video file, 133 setting color, 119 app-defaults directory, 170 APP1-APP63 (application function keys), hex value, 82 Application exiting base form, 36 initialization, 8 Application behavior changing default, in Windows, 156

changing the default, 20 options in Motif, 174 variables for controlling, 21–38

Application function keys (APP1–APP63), hex value, 82

Application messages, 45 adding, 48–49 header file, 53

Area attributes assigning, 116–117 defined, 115 removing, 117

Area graphics, setting, 118

AREAATT keyword (video file), 116-117

ARGR keyword (video file), 117

Arithmetic commands, 96, 98

Arrow keys hex value, 79 setting horizontal movement, 22 setting vertical movement, 23 wrapping behavior, 24

ARROWS keyword (video file), 130

ASCII extended control codes, 93 table of mnemonics and hex values, 83

ASGR keyword (video file), 107, 118 parameters, 118

Auto-wrap, setting margin, 113

В

BACK key (backtab), hex value, 79 Background color, resource in Motif, 172 Backslash in messages, 44 inputting, 92 Backspace, video file entry, 113 Base form, exiting, 36 Base window, 174

Basic colors defining in Motif, 171 defining in Windows, 155–156 keywords, 140 Behavior, variables for controlling, 21 Bell, setting in messages, 59 BELL keyword (video file), 131 BIOS flag (video file), 110 Bit-mapped attributes, 122 BKSP key (backspace), hex value, 79 BOFD key (beginning of field), hex value, 80 BOLN key (beginning of line), hex value, 80 Border keywords, 108, 128-130 limiting attributes, 130 setup variables, for menu bars, 32 styles, specifying alternate, 128-130 zoom window, 30 BORDER keyword (video file), 128-130 BOTTOMRT keyword (video file), 109 BOX keyword (video file), 130 BRDATT keyword (video file), 130 Buffer, setting size of output, 109 BUFSIZ keyword (video file), 109

С

CA, case interface message tag prefix, 44 Carriage return in message. *See* %N video file entry, 112 Case sensitivity, filenames, 33 CBDSEL keyword (video file), 131 CBSEL keyword (video file), 131 CBSEL keyword (video file), 131 Century specification, 36 CHAR_VAL_OPT, 36 Character JAM, setting line and box style in cmap file, 145 Character set 8-bit translation, 85, 125 graphics, 125–128

Character-sequence, defined, 77

Class name (Motif) application, 170, 178 field widgets, 180 menu widgets, 183 screen widgets, 179 widget, 178-184 CLICK_TIME, 24 CLR key (clear all), hex value, 79 cmap. See Configuration map file CMFLGS keyword (video file), 112 CMSG keyword (video file), 124 COF keyword (video file), 114 COLMS keyword (video file), 109 COLOR keyword (video file), 118-119 Color palette defining colors in Motif, 171 defining colors in Windows, 155 Color properties aliasing colors, 138-142 basic colors. See Basic colors display attributes, keywords, 140 highlighted colors, in Windows, 139 JAM basic colors, keywords, 140 Motif resources for overriding, 172 scheme. See Scheme Color terminal, display attributes in messages, 58 Command line, Motif, 177-188 bg switch, 178 fg switch, 178 name switch, 170 ownColormap switch, 178 Comments in key translation files, 77 in message file, 45 Compose characters, 85 Compose key, 85, 126 COMPRESS keyword (video file), 132 CON keyword (video file), 114 Configuration converting message files, 49-52 converting terminfo/termcap to video file, 101-102 converting video files, 104-105 Index

Configuration directory, contents, 3 Configuration files, required by JAM, 2 Configuration map file aliasing colors, 138-142 colors section, 138-142 object specification keywords, 143 scheme section, 142-145 screen editor section, 140-142 Configuration variables application-specific, 16-18 defined, 6 for setting path, 18 types, 13-14 Control characters, entering, 93 Control flow, commands, 96, 99 Control string binding to function key, 25 case sensitivity for filename searches, 33 Controlling input, variables for, 21-38 Conversion utilities key translation files to binary (key2bin), 86-87 message files to binary (msg2bin), 49-52 setup files to binary (var2bin), 10-11 video file to binary (vid2bin), 104-105 CUB keyword (video file), 113 parameters, 94 CUD keyword (video file), 113 parameters, 94 CUF keyword (video file), 113 parameters, 94 CUP keyword (video file), 114 parameters, 94 CURPOS keyword (video file), 132 Currency format, 66-70 default entries in message file, 68 Cursor appearance keywords, 107, 114-115 restoring, 115 saving, 115 setting, 22, 114 switching to/from system style, 114 behavior in groups, 34 defining gray/white keys, 110

movement defining, 22–24 setting video display, 113 position displaying, 132 keywords, 106, 112–114 restoring, 115 saving, 115 setting absolute, 114 setting absolute position (CUP), 114 specifying style of, 110 turning on/off. See CON/COF keyword

Cursor attributes. See Cursor, appearance

Cursor backward. See CUB keyword

Cursor down. See CUD keyword

Cursor forward. See CUF keyword

Cursor up. See CUU keyword

CUU keyword (video file), 113 parameters, 94

D

DA_CENTBREAK, 36 DARR key (down arrow), hex value, 79 Data compression, specifying for Jterm, 132 Database columns, setting number of, 109 Database message tag, 44 Date/time format customizing, 60–66 defaults, 61–62 for non-English JAM, 65 for non-English applications, 64 literal format for @date calculations, 66 tokens, 62–63 DDE

hot links, specifying in initialization file, 160 links, specifying in initialization file, 160

Decimal places, setting JPL default, 36

Decimal symbol, setting default, 69-70

DECIMAL_PLACES, 36

Defaults setting for Motif, 169-188 setting for Windows, 153-167 Defining keys, 83-86 Delayed write, 37 DELE key (delete character), hex value, 79 DELL key (delete line), hex value, 79 Display attributes as parameters, 21 defaults, assigning, 21 keywords, 140 keywords for video, 107 setting for zoom window borders, 30 in messages, 57-60 in status line, 26, 57 video attribute handling types, 115-124 video attributes ANSI terminals and, 120 combining, 121 for grayed menu items, 120 for message line, 125 for onscreen or area, 116-117 latch attributes, 120-123 video file keywords, 115-124 DM, database message tag prefix, 44 Drawing area, 179 Driver, keywords, 108, 132 Drop shadows

on character–mode screens, 33 setting emphasis, 119

DW_OPTIONS, 37

Ε

ED keyword (video file), 112 Editor, setting, 17 Eight–bit character set, 93 EL keyword, 112 EMOH key (go to last field), hex value, 79 EMPHASIS, 33 Emphasis style, defining, 33

EMPHASIS_KEEPATT keyword (video file), 119 EMPHASIS_SETATT keyword (video file), 120 EMSGATT, 27 ENTEXT_OPTION, 37 Environment variables, 6, 14-18 EOFD key (end of field), hex value, 80 EOLN key (end of line), hex value, 80 ER_ACK_KEY, 28, 80 ER_KEYUSE, 28 ER_SP_WIND, 28 Erase display command. See ED keyword Erase line command. See EL keyword Erase window command. See EW keyword Erasure command keywords (video file), 106, 112 Error acknowledgment key and space bar, 80 defining, 28 Error messages, acknowledgment, 28-39, 59 Escape sequence, for setting cursor style, 109 EW keyword (video file), 112 parameters, 94 EXIT key, hex value, 79 EXPHIDE_OPTION, 37 EXT key (extend selection), hex value, 80 EXTD key (extend selection down), hex value, 80 Extended colors, aliasing colors, 138-142 Extended keyboard, 84 defining for MS-DOS, 110 Extensions. See Filename, extensions EXTU key (extend selection up), hex value, 80

F

F_EXTOPT, 34 F_EXTREC, 34 F_EXTSEP, 34

Index

F11/F12 function keys accessing, 110 defining, 85 FCASE, 33 FE_KEEPATTRS, 31 FE_SETATTRS, 31 FE SWATTRS, 31 FERA key (clear field), hex value, 79 FHLP key (screen help), hex value, 79 Field, decimal symbol, 70 Field exit, setting validation condition, 24 Filename case sensitivity, 33 extensions, setting defaults, 33 for key translation file, 75 setting default behavior, 33-34 specifying screen extension, 34 Flow control. See Control flow FM, message tag prefix, 44 Foreground color, resource in Motif, 172 formMenus, 176 Function keys hex value, 81 setting default behavior, 25

G

GA_CURATT, 34 GA_CURMASK, 35 GA_SELATT, 35 GA_SELMASK, 35 GRAPH keyword, in video file, 126 Graphics characters keywords, *108* supporting, 125–128 Graphics sets, defining, 126 Grayed menu items, setting emphasis, 119 Graying, inactive screens, 33 GRAYKEYS flag (video file), *110*

Group cursor attributes, 34 display attributes, 34–35 occurrence attributes, 35 specifying selection/deselection characters, 131

GRTYPE keyword (video file), 126 keywords, 126

Η

Hard reset. *See* RESET keyword Header file creating, 53–56 sample, 53 HELP key (field help), hex value, *79* HOME key, hex value, *79*

I

IN_BLOCK, 22 IN_ENDCHAR, 23 IN_HARROW, 22 IN_RESET, 23 IN_VALID, 24 IN_VARROW, 23 IN_WRAP, 24 IND_OPTIONS, 29 IND_PLACEMENT, 30 Indicator symbol keywords (video file), 108, 130-132 submenu in character JAM, 132 INIT keyword (video file), 109 undoing effects of, 111 Initialization application options in Motif, 174-177 options in Windows, 156 JAM, 8

Initialization file. See Windows initialization file

194

INS key (insert/overwrite), hex value, 79 INSL key (insert line), hex value, 79 INSOFF keyword (video file), 114 INSON keyword (video file), 114 International characters, supporting, 85 Internationalization 8-bit characters, 85, 125 alternate message files, 71 currency formats, 66 date/time formats, 65 decimal symbol, 69–70 supporting, 127 yes/no values, 71

J

JAM basic colors, keywords, 140 jam.ini, 154 sample, 161–167 jamdev message tag, 44 JM, message tag prefix, 44 jmain.c, 154, 170 JPL, choosing an editor, 17 Jterm, enabling data compression, 132 JX, message tag prefix, 44 jxmain.c, 154, 170

Κ

KBD_DELAY keyword, in video file, 111
Key
See also Key label; Key translation; Key translation file
classes for PC extended keyboards, 84
label. See Key label
logical, 77
defined, 73
hexadecimal values, 78–82
mnemonics, 78–82
translation. See Key translation
Key classes for PC extended keyboards, 84

Key label, displaying in messages, 59

Key translation, variable, 15

Key translation file, 73-88 accessing, 88 comments, 77 converting to binary (key2bin), 86-87 creating and modifying, 83-86 defining as SMKEY variable, 83 identifying for initialization, 15 memory-resident, 86 modifying, 83-84 multiple, 75 names of, 2 naming convention, 75 purpose, 74-75 syntax, 77-83 using alternate files, 87 key2bin, 86-87

error messages, 87 Keyboard assigning timing interval, 111 extended, 84 for MS–DOS, 110 logical, mnemonics and hex values, 79–82 more than one type, 87

Keytop. See Key label

L

Label. *See* Key label Label text display, setup variables, 30 LARR key (left arrow), hex value, 79 Latch attributes defined, 115 setting, 120–123 LATCHATT keyword (video file), 120–123 LDB identifying files for initialization, 18

screen functions and, 37

Library message tag, 44

Line drawing for boxes, 130 keywords, 108, 128–130

Index

Line feed, video file entry, 113 Line styles setting in cmap file, 145 table of style names, 145 LINES keyword (video file), 111 LINEWRAP flag (video file), 116 List box widget, enabling extended selection, 37 List command, for parameter indexing, 100 LISTBOX_SELECTION, 37 Local decimal symbol, 69 Locking shifts, 127 Logical key changing behavior at runtime, 84 changing mapping of, 84 defined, 74 mnemonics and hex values, 78-82 required by jamdev, 79 Logical keyboard, vs. physical keyboard, 75 LP key (local print) defining default, 35 hex value, 79 LSHF key (left shift), hex value, 79 LWRD key (previous word), hex value, 79

Μ

MARKCHAR keyword (video file), 131 MB_BORDATT, 32 MB_BORDSTYLE, 32 MB_DISPATT, 32 MB_FLDATT, 32 MB_HBUTDIST, 32 MB_KEEPATTRS, 31 MB_LINES_PROT, 32 MB_SETATTRS, 32 MB_SWATTRS, 32 MB_SYSTEM, 32 MDI frame, placement of window in, 157 Memory-resident key translation file, 86 user messages, 46 video file, 104

Menu formMenus resource, 176 in character-mode, 31 widget hierarchy in Motif, 183 Menu bar reserving space for, 32 submenu indicator, 132 Menu item indicator symbol, setting for character mode, 131 setup variables, 31 Message See also Error messages; Message file; Status line acknowledgment, 60 forcing, 28, 59 bell, 59 creating, 44 default display in status line, 26 in window, 26 display attributes in, 27, 57-60 hexadecimal codes for, 58 displaying forcing to window, 26 in window, 60-69 file. See Message file forcing to status line, automatic dismissal, 59 installing, 48 JAM messages, 44 key labels in, 59 line. See Status line multiple lines in, 60 setup variables, 26-28 status, formStatus Motif resource, 175 status line. See Status line text not visible, 27 Message file, 41-71

converting to binary (msg2bin), 49–52 creating, 48–49 identifying for initialization, 16 modifying, 47 multiple sections, 45 name of, 2

size, 45 svntax, 44-47 text. 44 translating, 47 using alternate, 71 variable, 16 Message tag prefix, uses, 48 MESSAGE_WINDOW, 26 MNBR (menu bar key), 79 Mnemonic, display attributes in menu items, 31 in widgets, 31 MODE0 to MODE6 keyword, in video file, 126-128 Modifying messages, 47 Monochrome terminal, 58 Motif common color names, 173-174 setting defaults, 169-188 Motif resource file, 169–188 application behavior options, 174 background, 178 baseWindow, 174, 178 class name, 170 colors, 171-174 focusAutoRaise, 177 foreground, 178 formStatus, 175 global resources, 177-188 introPixmap, 174 location, 170 names, 169-170 overriding colors, 172 ownColorMap, 178 restricted resources, 177 restricting resources to a screen, 180 sample, Pages, 184 syntax, 170 MOUS key (indicate mouse event), hex value, 79 MOUS_CRSR_ATTR, 24 MOUS_CRSR_CHAR, 24 MOUS_CRSR_MASK, 25 Mouse, supporting in JAM, 132 Mouse driver, specifying, 132

Mouse pointer, character JAM appearance and behavior, 24
MOUSEDRIVER keyword (video file), 132
MS–DOS, INIT keywords, *110*msg2bin, 49–52 errors, 50
msg2hdr, 53–56 errors, 54 options/arguments, 53 sample output, 53
MSGATT keyword (video file), 125 flags for, 125
MTGL key (toggle menu mode), hex value, 79

Multiple sections, in message file, 45

MULTISHIFT flag (video file), 110

Ν

NL key (newline) acting like XMIT, 38 hex value, 79

No auto tab, setting, 23

Non-locking shifts, 127

Nonprinting characters, 93

0

Occurrence, setting attributes, 35

OMSG keyword (video file), 124

Onscreen attributes defined, 115 setting, 116

ONSCREEN flag (video file), 116

Output commands, 97 specifying, 95

Index

Ρ

Padding commands, 100 in termcap, 101 in terminfo, 101 Parameters, in video file. See Video file Path, 18 PC extended keyboard, 84 Percent commands arithmetic, 96, 98 for changing parameters, 99 for control flow, 96, 99 for parameter sequencing, 96, 98 for specifying output, 95, 97 for stack manipulation, 96, 98 for terminal delays, 100 in video file, 94-102 Percent escapes, in message file, 57 PF1-PF24 (function keys). See Function keys Portability, aliasing colors, 138 Prefixes, in message file, 44 Print file, setting system command, 35 Protected modes, 123

Q

QUIETATT, 27 Quotation marks, inputting, 92

R

RARR key (right arrow), hex value, 79
RCP keyword (video file), 115
REFR key (refresh screen), hex value, 79
Repeat character sequence See also REPT keyword setting, 111
REPMAX keyword (video file), 111
Repository, setting default pathname, 17

REPT keyword (video file), 111 parameters, 94

RESET keyword (video file), 111 Resource file. *See* Motif resource file REWRITE flag (video file), 116 rgb.txt, 173 RSHF key (shift right), hex value, *79* Runtime (JAM) message tag, 44 Runtime path, setting, 18 RWRD key (next word), hex value, *79*

S

SB OPTIONS, 30 Scan line, 110 Scheme defining in color map file, 142 object names for color mapping, 143 SCP keyword (video file), 115 SCR_KEY_OPT, 29 Screen See also Screen editor; Window character-mode attributes, 33 file extension, setting defaults, 34 focus, in Motif, 177 library. See Screen library resources, Motif, 180 setting number of lines, 111 startup, 37 widget hierarchy, Motif, 179 Screen editor, setting editor colors with cmap file, 140 Screen editor message tag, 44 Screen function data access, LDB vs. fields, 37 execution options, 37 Screen library, identifying for initialization, 17 SCREENWRAP flag (video file), 116 Scroll bars, slider characters, 131 Scroll indicators, setting position of, 30

198

Scrolling array indicator in video file. 130 placement, 30 setting, 29 setup options, 29-30 Set area graphics. See ASGR keyword Set graphics rendition. See SGR keyword Set latch graphics. See SGR keyword Setup file converting to binary (var2bin), 10-11 creating, 10 modifying, 10 name of, 2 sample, 39 specifying, 14, 16 syntax, 9 syntax of, 20 types of, 6-7 Setup variables changing, 20 defined, 6 defining, 20-21 delayed-write, 37 designating active screens, 33 display attribute keywords, 21 for character-mode screens, 33 for controlling behavior, 21-38 for cursor appearance, 22-38 for filenames, 33-34 for label text display, 30 for menus, 31 for message display, 26-28 for screen entry/exit processing, 37 for scrolling, 29-30 for shifting, 29-30 group attributes, 34-35 zoom, 29–30 Seven-bit character set, 93 SFTS key (shift scope), hex value, 79 SGR keyword (video file), 120, 123-124 parameters, 123 Shift indicators, setting position of, 30 Shifted function keys (SPF1-SPF24), hex value, 81 Shifting array. See Shifting field

Shifting field setting indicator, 29 setting indicator placement, 30 setup options, 29 specifying indicators for, 130 Slider characters, for scroll bars in character JAM, 131 SM, message tag prefix, 44 SM_CALC_DATE, setting default format, 66 SM_DECIMAL, setting default, 69 SMDICNAME, defined, 17 SMEDITOR, defined, 17 smerror.h, contents of, 42 SMFEXTENSION, 34 SMFLIBS, defined, 17 SMINICTRL, defined, 25 SMKEY, 83 defined, 15 smkeys.h, contents of, 74, 78-82 SMLDBNAME, defined, 18 SMLPRINT, defined, 35 SMMSGS defined, 16 setting alternate value, 65, 71 setting value of, 42 SMPATH, defined, 18 SMSETUP. 7 defined, 16 SMSGBKATT, 27 SMSGPOS, 26 SMTERM defined, 14 in Windows, 158 SMUSER, 158 SMVARS. 6 defined, 14 SMVIDEO, defined, 16 SMVIEWER, 18 Space bar, 80 Index

SPF1-SPF24 (shifted function keys). See Shifted function keys (SPF1-SPF24) SPGD key (scroll down page), hex value, 79 SPGU key (scroll up page), hex value, 79 Splash screen Motif, 174 Windows, 157 SPXATT keyword (video file), 124 Stack manipulation commands, 96, 98 STARTSCREEN, 37 Status line closing, 124 cursor position display, 132 display attributes, 124-125 force user to acknowledge, 28 formStatus resource, 175 keywords for video, 107, 124-125 location, setting in Motif, 175 opening, 124 setting display attributes, 26-28 position, 26 text attributes, 27 setup variables, 26-28 text not visible, 27 STEXTATT, 27 Submenu, indicator, 132 System decimal, defining symbol, 69 System menu, setup variable, 32

Т

TAB key acting like XMIT, 38 hex value, 79

TERM, 14 term2vid, 101–102

termcap, 90 padding in, 101 supporting, 91

Terminal assigning timing interval, 111 attributes, 115 bell, in message, 28, 59 initializing, 109 mapping input to output, 126 reset sequence, 111 timing interval, 100 type. See Terminal type visible bell, 131 Terminal type, setting, 14 Terminal-specific variable, 14 terminfo, 90 commands, not supported, 97 padding in, 101 Text, selection appearance, 25 Text editor, naming for JPL procedures, 17 Text widget, selection appearance, 25 Time format. See Date/time format Time-out delay, 132 Timing interval assigning to keyboard input, 111 setting with percent commands, 100 Title, for MDI window, 156 Toolbar enabling display, 26 setup variables, 26 TOOLBAR_DISPLAY, 26 Tooltip, enabling display, 26 TOOLTIP_DISPLAY, 26 TP, transaction monitor message tag prefix, 44 Transaction monitor interface message tag, 44 Translating message file, 47 messages, 43 physical keyboard, 73-88 substitution variables, 63, 68 TXT SELECT ATTR, 25 TXT_SELECT_MASK, 25 Type-ahead buffer, 28

U

UARR key (up arrow), hex value, 79 Underline display attribute, setting, 118 User messages. *See* Application messages UT, message tag prefix, 44 Utilities, file extensions, default behavior, 34 Utilities message tag, 44

V

Validation character-level, 36 setup options, 24 var2bin, 10-11 errors, 11 Variable configuration, 13-14 environment, 14-18 vid2bin, 104-105 errors, 104 Video file, 89-135 converting terminfo/termcap to, 101 converting to binary, 104-105 creating, 101-102 identifying for initialization, 16 international character support, 125-128 keyword summary, 105 names of, 2 parameter changing commands, 99 parameter sequencing commands, 98 for processing keywords, 91 parameters for keyword sequences, 93-101 sample ANSI terminal, 133 MS-DOS, 133 screen size entries, 111 syntax, 92-93 variable, 16 Viewing reports, 18 Visible bell, 131 VWPT key (viewport), hex value, 79

W

Wallpaper, for MDI window, 157 Widgets class names in Motif, 178-188 drawing area, 180 hierarchy, Motif, 178-188 base screen, 178-188 dialog box, 179 fields, 180 JAM screens, 179-188 menu bars, 183 win.ini, 160 Window displaying messages in, 60-69 placement in MDI frame, 157 Windows color definition, 155 control panel, 156, 160 setting defaults. See Windows initialization file WINDOWS flag (video file), 110 Windows initialization file 3D, 156

application behavior options, 156–167 colors, 155–156 FrameTitle, 156 JAM Colors, 155 location, 153 name, 154 sample, 161–167 setting defaults, 153–167 SMTERM, 158 splash screen, 157 syntax, 154–167 wallpaper for MDI window, 157 window placement in MDI frame, 157

WMODE (wordwrap toggle mode), hex value, 80

WNL (wordwrap newline), hex value, 80

Word wrapped text executing entry/exit procedures for each occurrence, 38 setting tab space, 38 WTAB (wordwrap tab), hex value, 80 WW_COMPATIBLE, 38 WWTAB, 38

Χ

XAPPLRESDIR, 171 Xdefaults, 171 XJam file, 170 sample, 184–188

XKEY flag (video file), 110

XMIT key (transmit) hex value, 79 making TAB and NL act like, 38 XMIT_LAST, 38

xrdb, 171

Y

Yes/No, setting default values, 71

Ζ

ZM_DISPLAY, 29 ZM_SC_OPTIONS, 29 ZM_SH_OPTIONS, 29 ZOOM key, hex value, 79 Zoom window

setting border attributes, 30 setting border style, 30 setup options, 29

ZW_BORDATT, 30

ZW_BORDSTYLE, 30