UNISYS

ALLY® Software Development Environment

System Manager's Guide

June 1988

Printed in U S America
UP–13765

Priced Item

Preface

This manual describes ALLY release 2.0.

This manual contains information for system managers on how to establish and maintain an ALLY environment on UNIX and MS-DOS systems. If you are running ALLY under another operating system, see the ALLY installation guide for your system.

The manual has nine chapters and four appendixes.

Chapter 1 explains ALLY's directory structure and files, ALLY environment variables, AFILE naming conventions, paths to help and error AFILEs, and compatibility among ALLY releases.

Chapter 2 describes how to invoke ALLY with commands and command files.

Chapter 3 describes each section of ALLY's Format File.

Chapter 4 describes the key-definition file and shows an example.

Chapter 5 discusses the terminal definition file and shows an example.

Chapter 6 explains the Terminal Definer utility.

Chapter 7 describes the printer definition file and shows an example.

Chapter 8 explains the Printer Definer utility.

Chapter 9 discusses how to route your printer output to different destinations.

Appendix A lists ALLY subdirectories and the files they contain in the UNIX environment.

Appendix B lists ALLY files in the MS-DOS environment.

Appendix C lists the mnemonics for all ALLY commands. These mnemonics are used in Macro Utility text files and in the key-definition file.

Appendix D contains a table of the ASCII character set with decimal, hexadecimal, and octal codes.

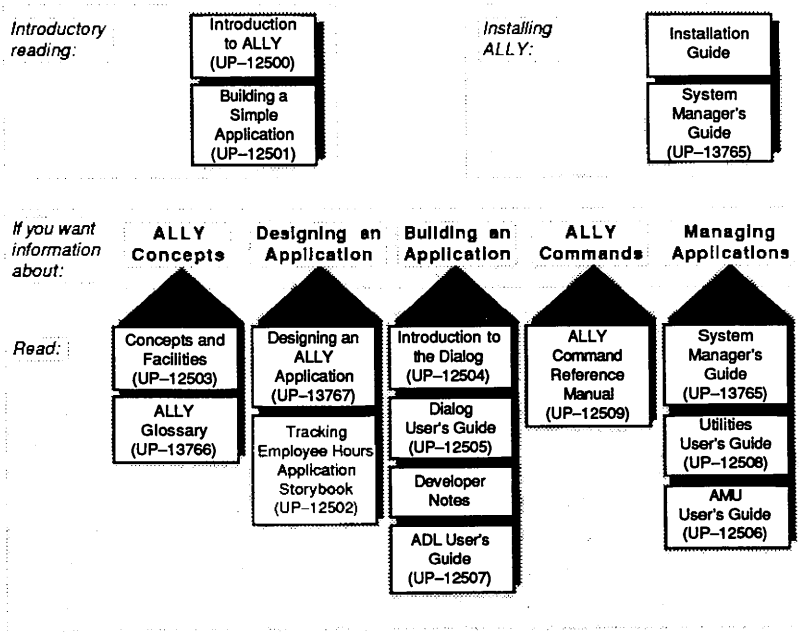
The ALLY Documentation—What to Read

The following illustration shows you how the ALLY documentation is organized.

If you want to develop ALLY applications, we suggest that you start by reading the ALLY system's introductory brochure, *Introduction to ALLY*. Then, you can build the application in *Building a Simple Application*.

If you want to install ALLY, you should read the installation guide for your system.

Note that the documentation for the runtime system of ALLY includes only the installation guide for your system, *ALLY Command Reference Manual*, and *AMU User's Guide*.



F002-0814-00

Conventions

You should read carefully the description of documentation conventions before reading this manual.

We use the following conventions in this manual:

Single quotes (' ')	Identify command names.
Boldface type (**bold**)	Highlights text you are to enter. Boldface is also used within command syntax statements.
Double quotes (" ")	Identify text strings within text sections. These strings are typically located in examples or as part of the prompts that ALLY sends to your display. Sometimes the exact content of a text string is affected by the traditional rules of punctuation. In these cases, we place the closing quotation mark at the end of the text string. For example, instead of: You see the prompt "Macro number:." We say: You see the prompt "Macro number:".
Brackets (<x>)	Enclose a specific key (x = key) to be typed. Angle-brackets are used in command syntax and key assignment lines. For example: <,> the "comma" key on the numeric keypad <Return> the "Return" key <Do> the "Do" key <F18> the "function" key 18 <2> the "2" on the numeric keypad
Empty brackets (<>)	Represent a form that requires input in menu path sequences, e.g., 1 2 <> 3 4.

Space (<Space>)	Represents a blank to be entered. A <Space> request asks you to type the space bar once.
Square brackets ([])	Enclose an argument for the command-line invocation of a utility. For example, **newprint** *[printer name] [output file].*

The key-definition files that are supplied with ALLY assign keys to several commands. For convenience, we assume that the following ALLY commands are assigned as shown.

Command	Key Assignment
'add new line'	<Return>
'up'	<↑>
'down'	<↓>
'right'	<→>
'left'	<←>
'define macro'	(See template)

The <↑>, <↓>, <→>, and <←> key assignments typically use a terminal's cursor-control keys.

End of Preface

Contents

Chapter 1. The ALLY Environment

Chapter 2. Invoking ALLY

Chapter 3. The Format File

Chapter 4. The Key-Definition File

Chapter 5. The Terminal Definition File

Chapter 6. The Terminal Definer Utility

Chapter 7. The Printer Definition File

Chapter 8. The Printer Definer Utility

Chapter 9. Managing Printer Output

Figures

Tables

Chapter 1
The ALLY Environment

Introduction

This manual contains information for system managers on how to establish and maintain an ALLY environment under UNIX and MS-DOS. If you are running ALLY under another operating system, see the ALLY installation guide for your system.

This chapter describes the following aspects of establishing and maintaining ALLY's working environment:

- the structure and contents of the ALLY system's directories
- location of important files
- environment variables
- AFILE naming conventions
- paths to help and error AFILEs
- ALLY's compatibility policy

The ALLY Directory Structure

Each ALLY release is stored in a directory named ally*x_yz*, where *x_yz* is the release number. For example, the directory for ALLY Release 2.00 is named "ally2_00." The structure of the ALLY directory depends on the operating system you are using. The next two sections describe the directory structure for UNIX and MS-DOS.

The ALLY Directory Structure Under UNIX

When you install ALLY Release 2.00 on a UNIX system, ALLY creates several subdirectories in the "ally2_00" directory. These subdirectories are listed in Table 1-1.

Table 1-1. Subdirectories in the ALLY Directory

Subdirectory	Contents
/afiles/	Subdirectories that contain AFILEs that ALLY uses
/allyexe/	ALLY executable files
/bin/	Operating-system command files to invoke ALLY facilities
/formats/	Format File used by ALLY utilities
/objects/ *	Compiled sample source code used by an external link that allows ALLY to communicate with a high-level language
/printers/	Printer description files
/src/ *	A subdirectory that contains sample source code used by an external link that allows ALLY to communicate with a high-level language
/term/	Terminal definition files

* Development systems only

Figure 1-1 shows the structure of ALLY's top-level directory under UNIX.

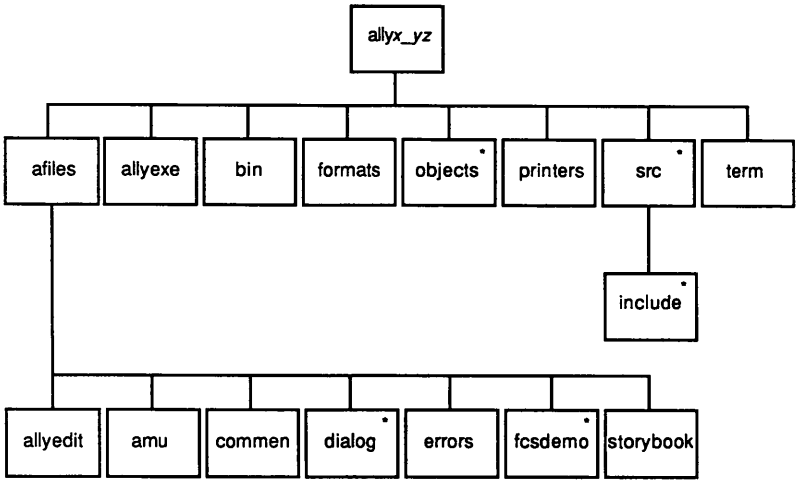


Figure 1-1. ALLY Directory Structure Under UNIX

The files in these UNIX subdirectories are listed in Appendix A.

The ALLY Directory Structure Under MS-DOS

On MS-DOS systems ALLY creates only one directory. This directory is installed by default in the C: drive. The "\ally2_00" directory contains the files that are listed in Appendix B.

ALLY Environment Variables

You can assign the names of files used by your ALLY application to operating-system environment variables. When an application executes, ALLY looks for the variable's value in the operating system's environment variable assignments.

Environment variables allow you to specify short variable names instead of longer path names. It is easier to change an environment variable's value than it is to change the value in an AFILE. Environment variables can also make your applications more transportable across operating systems by minimizing changes due to syntax differences.

Table 1-2 describes three operating-system environment variables that ALLY uses.

Table 1-2. ALLY Environment Variables

Variable	Value
ally	The path, or search list, to the top-level ALLY directory
TERM	The name of a terminal type (e.g., svt1220)
allyprinter	Overrides the default printer spooling device or queue name

The "ally" environment variable is the path to the ALLY directory. Therefore "ally" must be defined for each user process on UNIX systems and each time you reboot on MS-DOS. ALLY must also know what type of terminal you are using. ALLY can get this information from:

- the "TERM" variable setting on UNIX systems. On MS-DOS systems, ALLY uses the "pcterm" terminal description.
- an argument to an ALLY invocation command.

The "allyprinter" environment variable is described in Chapter 9.

Using Environment Variables

You can specify environment variables for the following parts of an application:

- help and error library AFILE names.
- utility file names (e.g., the terminal definition and key-definition files).
- data source file names for some access methods, including FX and C-ISAM. (See the developer notes for other access methods.)
- print file characteristics for form/report packets.
- global printer specifications (printer definition file, output file, spooling information).

Using ALLY environment variables involves three steps:

1) Displaying the values of environment variables.
2) Defining the value of environment variables with operating-system commands before your ALLY application executes.
3) Directing ALLY to use an environment variable by enclosing the name in braces in the Dialog.

Displaying Environment Variables

In standard UNIX installations, you can display the setting of an environment variable by typing the following command:

echo $*variable_name*

Table 1-3 lists the UNIX and MS-DOS commands used to display all of the environment variables that are defined for a user process.

Table 1-3. Commands to Display Environment Variables

Operating System	Command
UNIX Berkeley 4.x	printenv
UNIX System 5	env
MS-DOS	set

Defining Environment Variables

Table 1-4 lists UNIX and MS-DOS commands used to define an environment variable.

Table 1-4. Commands to Define Environment Variables

Operating System	Command
UNIX C-shell	setenv *variable value*
UNIX Bourne shell	*variable=value* export *variable*
MS-DOS	set *variable=value*

The MS-DOS "set" command defines a variable for the duration of a PC session. If you include the "set *variable=value*" command in an "autoexec.bat" file, the variable is automatically redefined whenever you reboot the system.

UNIX users can place the appropriate commands in the ".cshrc" (C-shell) file or ".login" (Bourne shell) file in their home directories.

After you have defined an environment variable's value with operating system commands, you direct ALLY to find an environment variable by enclosing the name in braces during a Dialog session (Figure 1-2).

Changing an Environment Variable Name

Suppose you want to run your application with a Format File in a different directory. You can do this by assigning an environment variable, for example "local," that contains the path to the new Format File. To use this Format File, type "{local}" as part of the name (Figure 1-2).

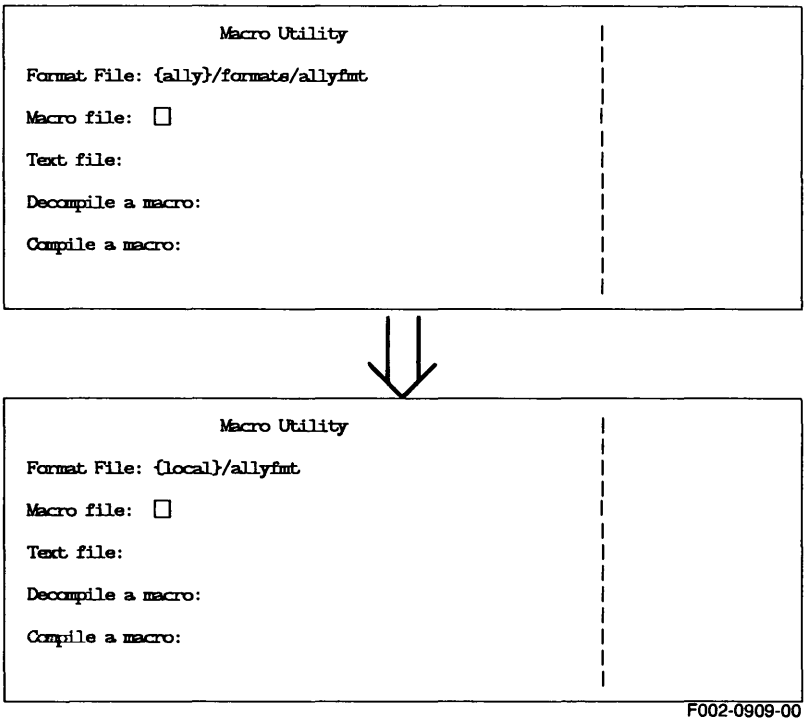


Figure 1-2. Changing an Environment Variable Name

AFILE Naming Conventions

Table 1-5 shows the conventions that ALLY uses to name different types of AFILEs.

Table 1-5. AFILE Naming Conventions

AFILE Type	Extension
Application	.a
Error message	.e
Help message	.h

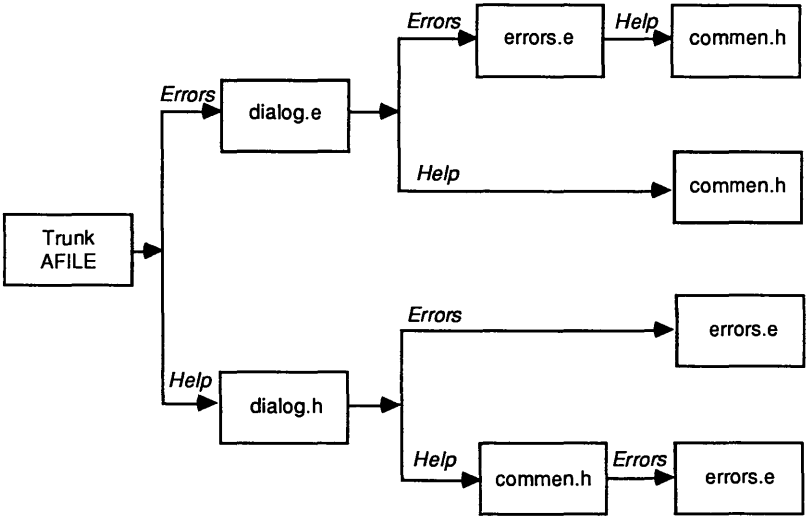
Using these conventions, here are some examples of how ALLY AFILEs are named:

amu.a	AFILE that contains the Application Maintenance Utilities (AMU) application
amu.h	AFILE that contains help messages for the AMU's forms and menus
amu.e	AFILE that contains error messages for AMU errors
commen.h	AFILE that contains help messages for command menus
errors.e	AFILE that contains error messages for general application-execution errors

Paths to Help and Error AFILEs

All ALLY applications can access a collection of general help and error messages. These messages are grouped in AFILEs called library AFILEs.

When creating an application, you start with an AFILE "skeleton," called a trunk AFILE. The Dialog creates the trunk AFILE for you when you specify a new AFILE name. A trunk AFILE contains default paths to Dialog-specific help and error AFILEs. These AFILEs, in turn, point to ALLY's general help and error AFILEs (see Figure 1-3). Note that every AFILE, including error AFILEs, can have both an error AFILE and a help AFILE.



ALLY's error AFILE for the Dialog is "dialog.e"

ALLY's general error AFILE is "errors.e"

ALLY's help AFILE for the Dialog is "dialog.h"

ALLY's help AFILE for command menus is "commen.h"

F002-0830-00

Figure 1-3. Paths to Help and Error Files in Trunk AFILEs

Changing Paths to Message AFILEs

You can create your own help message and error message AFILEs to provide messages specific to your application. Your application should point to its specific help and error AFILEs. These AFILEs should point to ALLY's general help and error AFILEs.

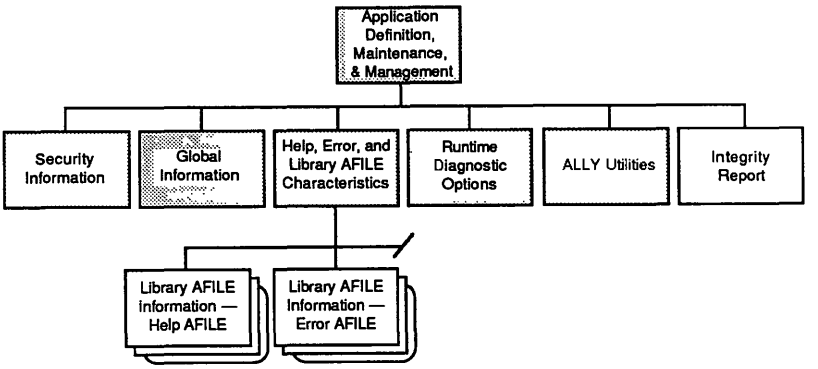
There are two ways to replace default paths with paths to user-defined message AFILEs:

1) With Dialog forms in the "Maintaining and Managing Applications" branch.

2) With special directives in a text file that the AFILE Message Builder utility uses to build a message AFILE.

Changing Paths with the Dialog

You can change paths to library AFILEs with application-specific paths by using two Dialog forms in the "Maintaining and Managing Applications" branch (Figure 1-4).



F002-0901-00

Figure 1-4. Changing Paths with the Dialog

Use the *Library AFILE Information—Error AFILE* form to change the path to your error AFILE. Use the *Library AFILE Information—Help AFILE* form to change the the path to your help AFILE. You can use these Dialog forms to change paths to message AFILEs in your application's AFILE or any of its message AFILEs. Refer to the *Dialog User's Guide* (UP-12505) for more information.

Changing Paths with Directives

The AFILE Message Builder utility creates a message AFILE from a text message file. Unless you change the paths, this message AFILE is linked to ALLY's general helps and errors.

When you build an error or help AFILE for your application, you can include two special directives in the message text file. These directives signify the path and file name to additional error and help AFILEs. If you do not include these directives, your message AFILE is automatically linked to ALLY's general help and error AFILEs. These directives are:

efn	The path and file name of an error AFILE
hfn	The path and file name of a help AFILE

Refer to the *Utilities User's Guide* (UP-12508) for more information about the AFILE Message Builder.

Paths to Application Message AFILEs

Figure 1-5 shows the paths when an application has its own help and error AFILEs.

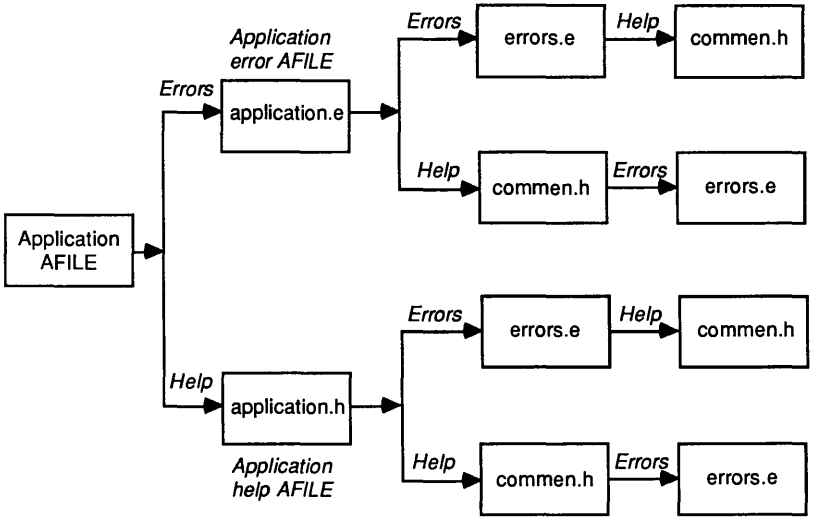


Figure 1-5. Paths to Application Help and Error AFILEs

If the paths to the library AFILEs are not complete, ALLY cannot find the message text when an error condition occurs or a user asks for help. Instead of message text, ALLY displays for:

errors	An internal ALLY number associated with the ALLY error condition
helps	An error message that ALLY cannot find the message text

Compatibility Among ALLY Releases

Parts of ALLY applications, such as AFILEs and the Format File, have version numbers that ALLY checks. The ALLY installation guide for your system lists specific version numbers that are valid for your ALLY release.

ALLY supports two types of compatibility:

Upward compatibility	Any new ALLY release can run any application AFILE built with an older ALLY release.
Downward compatibility	AFILEs created with new ALLY releases will execute under some older ALLY releases.

Upward Compatibility

When you execute the Dialog on an older application, ALLY asks you if you want to update the AFILE to the newer version number. If you respond by typing "Y<Return>," the AFILE is upgraded. You can also upgrade an AFILE with the AFILE Migrator utility or by compacting it with the AFILE Compactor utility.

NOTE: Once the AFILE is upgraded, you may not be able to execute it with an older ALLY release. Therefore, you may want to make a back-up copy of the old version of an AFILE before converting it..

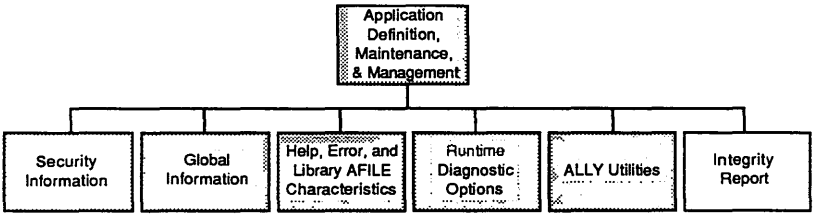
You can transport applications from one operating system to another even if both systems are not running the same ALLY release. However, if you transport an AFILE between different ALLY releases that do not both support the data's access method, you may have to:

- redefine the DSD
- transport the data with the Data Migrator utility

Downward Compatibility

The AFILE Migrator utility will provide downward compatibility for releases after ALLY version 2.00. That is, you can use a release 2.+ AFILE Migrator to revert an AFILE to a lower release number.

If you transport an AFILE to a lower release number, you should check the integrity of the whole application. The integrity report will list any elements (e.g., AFILE item types or ADL functions) that are not supported in the lower version AFILE. Figure 1-6 shows the location of the Dialog form (*Integrity Report*) that you use to check your application's integrity.



F002-0902-00

Figure 1-6. Location of the Dialog Form for Integrity Reports

End of Chapter 1

Chapter 2
Invoking ALLY

Introduction

This chapter describes two methods for invoking ALLY and the ALLY utilities from your operating system. You can invoke ALLY:

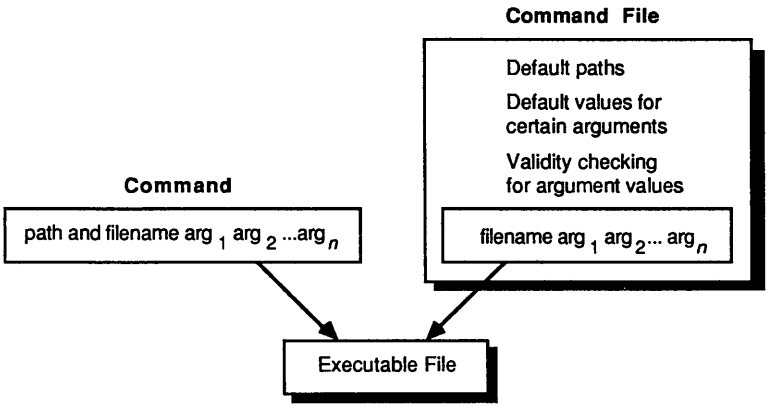
- with a command
- with a command file

A *command* invokes an ALLY executable file directly. A command consists of the path and file name of an executable file, followed by required arguments.

A *command file* also invokes an executable file. In addition, command files check the arguments for validity, provide a default path to the executable file, and provide default values for some of the command's arguments. Because command files provide default values for arguments, they require less typing than commands do.

Arguments to both commands and command files are positional. Specify "none" as a position "placeholder" for arguments that do not have a value (e.g., password or symbol table file).

Figure 2-1 shows the relationship of commands and command files to the executable files that they invoke.



F002-0878-00

Figure 2-1. Commands and Command Files

Using Commands

When you invoke ALLY and its utilities with a command:

- All of the command arguments are positional and required.
- The executable file name, and any other argument that specifies a file, must include the file's path.

On UNIX systems, these executable files are in the "/allyexe/" subdirectory. On MS-DOS systems, the executable files have an extension of ".exe" and are in the "ally*x_yz*" directory (*x_yz* is the release number).

Using Command Files

When you invoke ALLY and its utilities with the command files that are shipped with your ALLY release:

- the arguments are positional. However some have defaults and are not required. Because the arguments are positional, you must specify a value for each argument that precedes the last one you want.
- The default values for ALLY command files specify that AFILEs have internal symbol tables and are not password-protected (if you do not provide values for these arguments).
- A command file provides a default path to the executable file.
- A command file that invokes a utility uses the Format File installed with your ALLY release.
- A command file provides a default value for the terminal description file, if the executable file requires one.
- On UNIX systems, the command files are in the "/bin/" subdirectory and are written in the Bourne shell syntax. On MS-DOS systems, the command files have an extension of ".bat" and are in the "ally*x_yz*" directory.
- The "ally" environment variable must be defined before you invoke command files.

Invoking ALLY

This section describes the two versions of the ALLY execution system, and then shows the syntax of the commands and command files that are shipped. The commands show the executable file names; note that they end with an "x." You can type a command file name followed by the single argument "help" to display information about the arguments that have default values and are thus optional.

There are two versions of the ALLY execution system: the development system and the runtime system.

The development system runs:

- the Application Developer's Dialog
- the ALLY utilities
- any ALLY application
- the ALLY Text Editor

The runtime system:

- runs any application built with the Dialog
- does not run the Dialog; therefore users cannot change an application
- runs the ALLY Text Editor
- includes a set of utilities, called the Application Maintenance Utilities (AMU), that allows users to tailor applications for different terminals and printers

Invoking an Application

The following command runs an AFILE.

allyrunx *[terminal description file] [AFILE] [macro file] [entry point] [debug log]*

The arguments for this command are:

terminal description file	is the name of the file that contains information about your terminal.
AFILE	is the name of the AFILE that you want to run.

macro file	is the name of a file, defined in a previous ALLY session, that contains ALLY keystrokes. This allows you to reuse macro files, rather than having to define macros in each session. Specify "none" if you do not want to include a macro file.
entry point	is the name of the entry point in the application. By convention, "MAIN_TASK" is the name of the default entry point.
debug log	is the name of the file to which you want the AFILE debugger messages written. The debugger selectively traces the execution of ALLY actions, tasks, and data transfers. Specify "none" if you do not want a debug log.

The following command file runs an AFILE.

ally *[AFILE] [terminal description file] [entry point]*

Only the first argument is required, because the other two have defaults.

Invoking the Dialog

Because the Dialog is an ALLY application, the arguments to run the Dialog are the same as the arguments shown previously for running any application. The Dialog's AFILE name is "dialog.a."

The following command invokes the Dialog:

allydevx *[terminal description file]* *[***dialog.a***]* *[macro file]* *[***MAIN_TASK***]* *[debug log]*

Type the name of the Dialog AFILE, preceded by its path, for the second argument. Note that the entry point must be "MAIN_TASK."

The Dialog command file is:

dialog *[terminal description file]*

The terminal description file argument is optional because the default value is the same as your "TERM" environment variable.

Invoking the Application Maintenance Utilities

The command to invoke the Application Maintenance Utilities (AMU) is:

amux *[terminal description file]* *[***amu.a***]* *[macro file]* *[***MAIN_TASK***]* *[debug log]*

Type the name of the AMU AFILE, preceded by its path, for the second argument. Note that the entry point must be "MAIN_TASK."

The AMU command file is:

amu *[terminal description file]*

The terminal description file argument is optional.

Invoking ALLY Utilities

This section shows the syntax of the commands and command files that are shipped with the ALLY utilities. The commands and the command files are listed below.

See Chapter 6 and Chapter 8 for details about the arguments for the Terminal Definer and Printer Definer utilities. Refer to the *Utilities User's Guide* (UP-12508) for details about arguments for the other utilities' command files. You can type a command file name followed by the single argument "help" to display information about the arguments that have default values and are thus optional.

The AFILE Compactor

The AFILE Compactor command is:

compactx *[Format File] [input AFILE] [output AFILE] [symbol table file] [password] [options]*

The AFILE Compactor command file and its arguments are:

acompact *[input AFILE] [output AFILE] [symbol table file] [password] [options]*

The last three arguments (symbol table file, password, and options) have default values.

The AFILE Merger

The AFILE Merger command is:

mergex *[Format File] [master AFILE] [master AFILE password] [second AFILE] [second AFILE password] [output AFILE] [symbol table file] [output AFILE password] [options]*

The AFILE Merger command file and its arguments are:

amerge *[master AFILE] [master AFILE password] [second AFILE] [second AFILE password] [output AFILE] [symbol table file] [output AFILE password] [options]*

The last argument (options) has a default value.

The AFILE Message Builder

The AFILE Message Builder command is:

newmsgx *[Format File] [AFILE] [symbol table file] [password] [Dialog AFILE] [options] [text file(s)]*

The AFILE Message Builder command file and its arguments are:

newmsg *[AFILE] [options] [text file(s)] [password] [symbol table file] [Dialog AFILE]*

Note that the order of the arguments in the command is different from the order in the command file. This is because the command file has default values for the password, symbol table file, and Dialog AFILE; thus these arguments are optional.

The AFILE Migrator

The AFILE Migrator command is:

amigratx *[Format File] [input file] [output file] [symbol table* or *password] [***r** or **w***] [***v***]*

The AFILE Migrator command file and its arguments are:

amigrate *[input file] [output file] [symbol table* or *password] [***r** or **w***] [***v***]*

The AFILE Script Writer

The AFILE Script Writer command is:

scriptx *[Format File] [input AFILE] [output text file] [password] [options]*

The AFILE Script Writer command file and its arguments are:

ascript *[input AFILE] [output text file] [password] [options]*

The last two arguments (password and options) have default values.

The Data Migrator

The Data Migrator command is:

dmigratx *[terminal description file] [Format File] [AFILE] [password] [DSD name] [text file] [***r** or **w***]*

The Data Migrator command file and its arguments are:

dmigrate *[AFILE] [password] [DSD name] [text file] [***r** or **w***]*

Note that the command file does not require the terminal description file argument.

The Macro Utility

The Macro Utility command is:

mmigratx *[Format File] [input file] [output file] [***c** or **d***]*

The Macro Utility command file and its arguments are:

mmigrate *[input file] [output file] [***c** or **d***]*

The Printer Definer

The Printer Definer Command is:

newprntx *[Format File] [printer name] [output file] [printer definition file]*

The Printer Definer command file and its arguments are:

newprint *[printer name] [output file]*

Note that the command file does not require the printer definition file argument.

The Terminal Definer

The Terminal Definer command is:

newtermx *[Format File] [terminal name] [output file] [key-definition file] [terminal definition file] [option]*

The Terminal Definer command file and its arguments are:

newterm *[terminal name] [output file] [key-definition file] [option]*

The last argument (option) has a default value. Note that the command file does not require the terminal definition file argument.

End of Chapter 2

Chapter 3
The Format File

The Format File contains the message text that the ALLY utilities produce. Error text and status messages are stored in the Format File and in ALLY's error AFILE (errors.e). This simplifies translating ALLY applications to other languages, because you need to edit only the text in the Format File and the files used to build the "errors.e" AFILE.

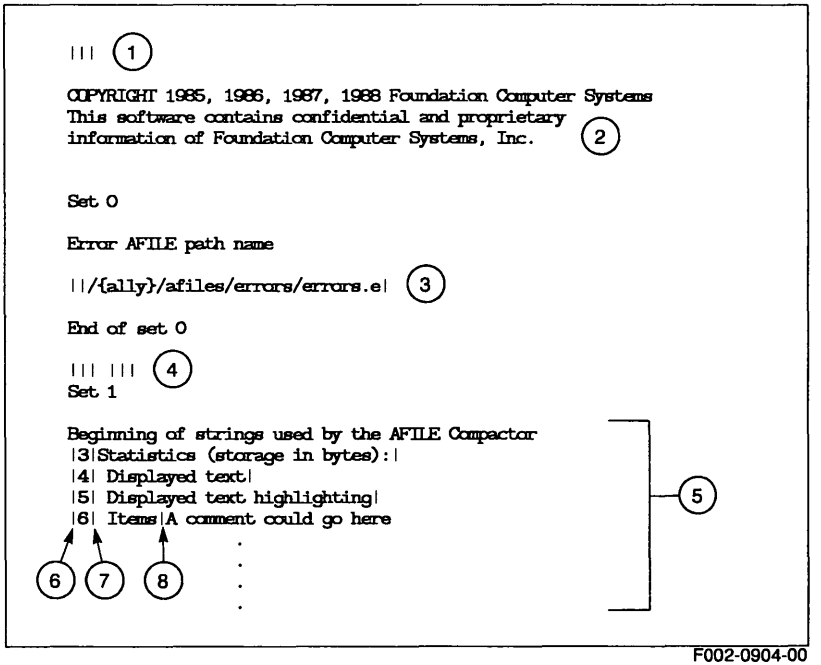
When ALLY is shipped, the Format File is named "allyfmt." On UNIX systems, the Format File is in the "/formats/" subdirectory. On MS-DOS systems, the Format File is in the "ally*x_yz*" directory (*x_yz* is the release number).

Before you attempt to make any changes to the Format File, make a copy so that you will always have a valid version. Even a change that appears innocuous (e.g., the addition of a blank character) can prevent the utilities from functioning properly.

Format File entries are associated with entry numbers and are separated by delimiter characters. You should never change the entry numbers, their positions, or the characters that delimit them. Changes to the text strings used in reports that the utilities produce are necessary only when you are translating the application to another language.

Syntax of the Format File

Figure 3-1 shows the beginning of a Format File as it is shipped with ALLY.



F002-0904-00

Figure 3-1. Syntax of a Format File

1) The first three characters in a Format File are the delimiter characters. The vertical bar character (|) is the delimiter in the Format File that is shipped with ALLY.

   All three delimiters can be the same or different characters. Do not use delimiter characters that you might use as text in entries or comments.

2) Anything after these initial three delimiter characters, up to the next start delimiter, is treated as a comment.

3) The Format File is made up of several sections (called sets) that contain entries. The entry for the first section (Set 0) must be the path and file name to ALLY's errors AFILE, shown here for UNIX systems. The utilities need this path in order to produce the text for error conditions that can arise.

4) Each set begins with the three delimiter characters and ends with the same three delimiter characters that began it. By convention, the three delimiters that end a set and the three delimiters that start the next set are on the same line.

5) Each entry contains:

    6) the first delimiter character, followed by an entry number.

    7) a delimiter character after the entry number, indicating the start of the entry.

    8) a delimiter character after the entry, that indicates the end-of-entry and the start of an optional comment.

The next characters can be either comments or the delimiters for the next section. (Space or text between two sets of delimiters is treated as a comment.) The delimiters do not have to be the same as those of the previous section.

Although you can change the entry text, you cannot change the entry number or the item that is associated with the number. The parts of a Format File that you must not change are:

- delimiters, once you have defined them
- entry numbers
- any specials symbols that are not in a comment

Table 3-1 lists the sections (sets) of the Format File that are provided with ALLY and the utilities that use them.

Table 3-1. Sections of the Format File

Set Number	Used by
0	All utilities
1	The AFILE Compactor
2	The AFILE Script Writer
3	The AFILE Script Writer
4	The Terminal and Printer Definers
5	The AFILE Message Builder
6	(reserved)
7	The AFILE Merger
8	The Macro Utility
9	The AFILE Migrator
10	The Data Migrator
11	The AFILE Script Writer

AFILE Compactor Section

The AFILE Compactor uses set 1 of the Format File (Figure 3-2). Entries in set 1 provide the text for the statistics report that the AFILE Compactor produces.

```
Set 1

Beginning of strings used by the AFILE Compactor
|0|undefined |
|1|.|
|2|
|
|3|Statistics (storage in bytes):|
|4| Displayed text|
|5| Displayed text highlighting|
|6| Items|
|7| ADL streams|
|8| First block|
|9| Used|
|10| Empty|
|11| Total compacted file size|
|12| Character strings|
|13| Symbols|
|14| Back pointers|
|15|none|
                .
                .
                .
```

Figure 3-2. Sample AFILE Compactor Section

AFILE Merger Section

The AFILE Merger utility uses set 7 of the Format File (Figure 3-3). For the most part, the AFILE Merger entries provide the text for the messages that report global information differences in the two AFILEs being merged.

```
Set 7

Beginning of strings used by the AFILE Merger

|0|Character sorting sets are different
|1|none|
|2|SYSTEM_SECURITY|
|3|Difference in num_parse_chars
|4|A non-context was encountered when a context was expected
|5|Input AFILE 1 is not at the current AFILE version level|
|6|Input AFILE 2 is not at the current AFILE version level|
|7|One or more differences found in first blocks of input AFILEs
|8|Global variable conflict |
|9|Entry point conflict |
|10|**unused**|

*** Difference in first block detected by AMSCAF ***
|11|**unused**|
|12|Difference in machine_flags
|13|Difference in num_vm_buffers
|14|Difference in help_info|
|15|Difference in error_info|
|16|Difference in spool_device|
|17|Difference in printer_description
|18|Difference in print_file|
|19|Difference in global_menu_type
|20|Difference in path_separator
|21|Difference in prev_string|
|22|Difference in top_string|
|23|Difference in num_in_fmt_chars|
|24|Difference in num_out_fmt_chars|
|25|Difference in def_num_attr|
|26|Difference in def_char_attr|
|27|Difference in num_def_val|
|28|Difference in char_def_val|
|29|Difference in date_def_val|
|30|Difference in logon_set|
|31|Difference in arithmetic_flags|
|32|Difference in sec_retries|
|33|Difference in global_control_flags|
|34|Difference in menu_prompt_size|
|35|Difference in def_date_masks|
|36|Difference in date_strings|
|37|**unused**|
*** Messages related to integrity check ***

|38|Integrity errors found in AFILE

End of strings used by the AFILE Merger

||||||
```

Figure 3-3. Sample AFILE Merger Section

AFILE Message Builder Section

The AFILE Message Builder utility uses set 5 of the Format File. Set 5 entries:

- define defaults for message AFILE structures and values that are constant for most applications at a site
- define special strings, called directives, that control highlighting or signify the type, beginning, and end of each message
- provide default values for invocation command arguments, such as "none" for an external symbol table file or password
- provide special directives for paths to user-defined help and error AFILEs that override default paths to ALLY's helps and errors

When the Message Builder creates a message AFILE from an input text file, that AFILE has:

- error message text for each message in the text file that starts with the "e" directive
- help message text for each message in the text file that starts with the "h" directive
- embedded string text for each message that starts with the "s" directive
- legend text for each message that starts with the "l" directive

The text is displayed exactly as it is entered in the input text file, except that the directives and control characters (other than "tab" and "newline") are removed. It is a good idea to limit the length of all lines to eighty characters to eliminate difficulty when:

- working with text editors that have an eighty-column length limitation. (ALLYedit does not have this limitation.)
- transporting message files to different computer systems.

Each error message, help message, embedded string, and legend must be assigned the same number as the internal AFILE number of an event that calls the message. The message number (or numbers) is specified after the message start directive ("e," "h," or "l").

You can associate a message with more than one number, so that the same message can be used by many fields, menus, choices, or forms/reports. The number(s) must be the first characters after the start directive and must be specified on one line, unless a continuation character is present at the end of the first line. The continuation character is defined by entry 49, which is a back slash (\) in the Format File that is shipped with ALLY. Separate the message numbers with the character specified by entry 122, which is the message header number separator. Entry 122 is a space in the Format File that is shipped with ALLY. Do not specify duplicate message numbers.

Directives for Message Text Type

Set 5 entries 1 through 3, 67, and 124 are directives that identify text as either a help or error message, a legend, or an embedded string. Entry 67 is the directive for assigning a second-level help message (a help message for a help message). Entry 4 is the directive that signals the end of the message. These entries are shown in Figure 3-4.

The numbers that follow these message type directives are those that you have assigned as help, error, and legend numbers for parts of your application. Directive 34 specifies the default help number assigned to help and error messages that do not have an explicitly assigned message number.

```
|1|$e$|error header
|2|$e$|embedded string header
|3|$h$|help header
|4|$*$|end of message

|34|32764|form/report help number

|67|$help$|message help_number

|124|$1$|legend header
```

Figure 3-4. Set 5 Entries for Message Type

Directives for Paths to Error and Help AFILEs

The Format File provides default paths and file names to ALLY's general error and help AFILEs. The AFILE Message Builder uses these default paths when it builds message AFILEs. The default path to ALLY's general error AFILE is entry 83. The default path to ALLY's general help file is entry 84. Figure 3-5 shows entries 83 and 84 from Set 5 in the Format File for UNIX systems.

```
|83|{ally}/afiles/errors/errors.e error AFILE
|84|{ally}/afiles/common/common.h help AFILE
```

Figure 3-5. Format File Entries 83 and 84 from Set 5

You can change the default paths to user-defined error and help AFILEs by including special directives that override these defaults at the beginning of your text file for input to the Message Builder utility. This establishes paths from the application's help and error AFILEs to ALLY's general help and error AFILEs. The overriding directive for a user-defined error AFILE is entry 56. The overriding directive for a user-defined help AFILE is entry 63. Figure 3-6 shows entries 56 and 63 from Set 5 in the Format File for UNIX systems.

```
|56|$efn$||directive for user-defined error AFILE

|63|$hfn$||directive for user-defined help AFILE
```

Figure 3-6. Format File Entries 56 and 63 from Set 5

AFILE Migrator Section

The AFILE Migrator section of the Format File is set 9 (Figure 3-7).

```
Set 9

Beginning of strings used by the AFILE Migrator

|0|AFILE item name, AFILE item type, AFILE version number |
|1|DSD AFILE item name, access method number|
|2|**unused**|
|3|none| no external symbol table file
|4|,  |
|5|.|
|6|NONAME|
|7|The |
|8|version of the AFILE Migrator did not recognize the|
access method of the following DSD AFILE items.
The DSD AFILE items are from an AFILE with version number |
|9|For a description of the access method number, see the documentation
for the version of ALLY that supports version |
|10| AFILEs.|
|11|The AFILE contains the following unrecognized types of AFILE items.
For a description of the following item type numbers,
see the documentation for the version of ALLY that supports the AFILE
version number listed.|

End of strings for the AFILE Migrator

||||||
```

Figure 3-7. Sample AFILE Migrator Section

AFILE Script Writer Sections

The AFILE Script Writer sections of the Format File are:

- Set 2 Contains ALLY internal item names
- Set 3 Contains error messages
- Set 11 Contains control strings that govern Script Writer execution and the text for Script Writer output

You can change *only* the ASCII text strings in set 11 that are used to produce Script Writer reports. Figure 3-8 shows part of the Script Writer's section 11.

```
Set 11

Beginning of strings used by the AFILE Script Writer
                .
                .
                .
MENU|5
     $7$    ($10$)
     $547$$36$
     Developer comment:  $32$% $15$ $515$$36$
_____________________________________________________________________$33$%$37$
     No text defined for this menu%
_____________________________________________________________
     Menu window:  ($506$,$507$) to ($508$,$509$)
     Prompt area:  ($511$,$512$) to ($513$,$514$)$510$$51$
$550$Menu-Style Options Set:  $58$$0$
     - Numeric menu %$58$$1$
     - Function-key menu %$58$$2$
     - Cursor-roam menu %$58$$3$
     - Letter menu %$58$$4$
     - Word menu %$58$$5$
     - Roam highlighting %$58$$6$
     - Case sensitive %$58$$7$
     - Minimal match %$58$$8$
     - Jumping allowed %$58$$9$
     - Paths allowed %$58$$10$
                .
                .
                .
```

Figure 3-8. Sample AFILE Script Writer Section

Data Migrator Section

The Data Migrator uses set 10 of the Format File (Figure 3-9). These text strings produce the descriptions of the elements in a DSD.

```
Set 10

Beginning of strings used by the Data Migrator utility
                .
                .
                .
|12|), |
|13|) |
|14|MM/DD/YYYY HH24:MI:SS|
|15|continuing without date strings
|
|16|FLOAT|
|17|FIX|
|18|TRAILING BLANKS|
|19|Item pointer|
|20|String pointer|
|21|References 1 byte flag|
|22|References 2 byte flag|
|23|AFILE
|
|24|references bit |
|25| negative logic|
|26| positive logic|
|27|number format|
|28| width |
|29| precision |
|30|string format|
|31| length |
|32|date picture |
|33|Call type number|
|34|Aptr to an action|
                .
                .
                .
```

Figure 3-9. Sample Data Migrator Section

Macro Utility Section

The Macro Utility uses set 8 of the Format File (Figure 3-10). Most of these text strings are the names of ALLY commands.

```
Set 8

Beginning of strings used by the Macro Utility

|0|?|
|1|?|
|2|?|
|3|?|
|4|ABORTACTION|
|5|EXITACTION|
|6|ABORTTASK|
|7|EXITTASK|
|8|ABORTAPPL|
|9|EXITAPPL|
|10|REFRESH|
|11|KHELP|
|12|SHELL|
|13|SET_REPEAT_CNT|
|14|PRNTSCRN|
|15|PRNTVNUM|
|16|EXPANDWDW|
|17|COMPRESSWDW|
|18|MOVEWDW|
|19|DEFINEWDW|
|20|SCROLLWDW|
|21|EXPLODEWDW|
|22|RESIZEWDW|
|23|DEFMACRO|
|24|MACTOFILE|
|25|MACFMFILE|
|26|SAVEMACROS|
|27|LOADMACROS|
|28|EXEMAC0|
|29|EXEMAC1|
          .
          .
          .
```

Figure 3-10. Sample Macro Utility Section

Terminal and Printer Definer Section

The Terminal Definer and Printer Definer share set 4 of the Format File (Figure 3-11). Most of the text strings in set 4 describe problems that prevent the terminal or printer description files from being created.

```
Set 4

Beginning of strings used by the Terminal Definer and Printer Definer utilities


|0|WARNING: |
|1|ERROR  : |
|2|keyname |
|3| in line number |
|4| in keydef file |
|5|invalid octal character
|6|invalid ^Q or ^S in control sequence numbers |
|7| has incorrect format |
|8|execution stopping |
|9|key conflict |
|10| at lines |
|11| and |
|12| not found|
|13|terminal |
|14||
|15|line number|
|16| cannot open termdef database file |
|17| not found in termdef database |
|18|**unused**|
|19|cannot open output file |
|20|printer |
|21| not in printdef database |
|22| cannot open key definition file: |

End of strings used by the Terminal Definer and Printer Definer utilities

||||||
```

Figure 3-11. Sample Terminal and Printer Definer Section

End of Chapter 3

Chapter 4
The Key-Definition File

ALLY provides extensive support for keyboards. Because ALLY commands can be assigned to different keys or key sequences, ALLY and ALLY applications are as independent of keyboards as they are of terminals and printers.

A key-definition file assigns commands to terminal keys. The Terminal Definer utility uses the key-definition file to build a terminal-description file that defines the terminal to ALLY. There must be a key-definition file for each terminal on which ALLY executes. Key-definition files for several terminals are included in your ALLY release.

Command-to-Key Assignments

You can assign any ALLY command to any terminal key or keys. You make these key assignments in a key-definition file.

Each key on an asynchronous terminal sends one or more octal codes to the operating system. The terminal manufacturer determines the code or sequence of codes that a key transmits. The documentation supplied by the terminal manufacturer lists the octal code(s) transmitted by each key. You assign a command to a key or keys by specifying in the key-definition file the mnemonic for the command next to the octal code for one or more keys.

Certain keys are defined to be used in combination with another key; that is, you type one key and while pressing it, you type another key. The "Ctrl" key is often used in combination with another key in this fashion.

You can also assign commands to a key combination, or to a series of keys, where you type one key and then type another key. For example, suppose the command to refresh the terminal screen is assigned to the key series, <PF1> and <PF2>. To invoke the 'refresh' command, you first type <PF1> and then type <PF2>, instead of continuing to press <PF1> while you type <PF2>.

It is useful to understand all of the ALLY commands that can be invoked by key. A list of ALLY command mnemonics is included in Appendix C. You can refer to the *ALLY Command Reference Manual* (UP-12509) for a complete description of each ALLY command.

Syntax of a Key-Definition File

Because there are more ALLY commands than keys, ALLY provides key-definition files with the most commonly used commands assigned to keys. The remaining commands can be used through the Command Menus. However, you must assign the 'do' command to a key in order to access the Command Menus.

There are eight sections in a key-definition file. The first two are global sections that affect the remaining six local sections. Key assignments in a local section apply only to that section. Therefore, different commands can be assigned to the same key (or keys) from one local section to another. It is important to note that a key can have a different command assigned to it for different ALLY subsystems (e.g., menus, forms/reports, ALLYedit).

For example, <Return> can invoke the 'next field' command in forms/reports and the 'add new line' command in ALLYedit because the commands are in different local sections of the key-definition file.

The syntax rules for a key-definition file are:

- Each mnemonic must be followed by a colon (:).
- The octal code or code sequence for the key (or keys) that invokes the command must follow the colon.
- A line that begins with a semicolon (;) is a comment.
- Each section must terminate with an "at" sign (@) at the beginning of a line.

Figure 4-1 shows the format of a key-definition file. The Terminal Definer displays an error message if the key-definition file deviates from the correct format. A complete key-definition file is included at the end of this chapter.

```
;
; WINDOW COMMANDS
;
; Function      Octal Code              Key Assignment
; --------      ----------              --------------
;
winup:          033 133 101             ; up arrow
windown:        033 133 102             ; down arrow
winright:       033 133 103             ; right arrow
winleft:        033 133 104             ; left arrow
windone:        015                     ; carriage return
@

Mnemonic        Key assignment          Comment
```

Figure 4-1. Sample Entries in a Key-Definition File

Sections of a Key-Definition File

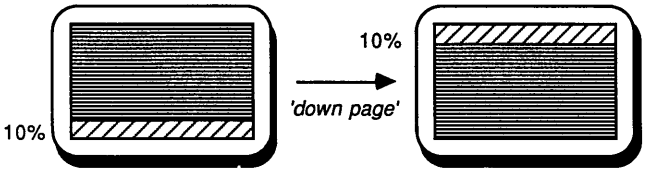
The sections of the key-definition file are arranged in the following order:

- global flag definitions and scroll percentage
- global commands
- text editing commands
- form/report commands
- window commands
- menu cursor roam commands
- menu function task commands
- menu prompt line terminators

Each section is briefly described in the following pages.

Global Flag Definitions and Scroll Percentage Section

The commands in this section are global to ALLY and are not active commands. Rather, they are flags and variables that specify initial settings for things such as case sensitivity for searching and the amount to scroll a page when editing. The OFF setting for these flags and variables is octal 000 and the ON setting is octal 001. The default for the scroll percentage variable is 012, which leaves ten percent of the screen on page scroll (Figure 4-2).



F002-0563-00

Figure 4-2. Scroll Percentage

Global Commands Section

The global commands in this section:

- invoke macros and help messages
- activate and manipulate tasks, including Command Menus
- abort and exit from an action, a task, and an application

They are called global commands because they can be used in any ALLY subsystem (text editor, image editor, form/report, etc.).

Text Editing Commands Section

These text editing commands apply whether you are editing a text file or a field in an ALLY application.

Form/Report Commands Section

ALLY forms and reports are unified and use the commands in this section.

Window Commands Section

These commands are for changing window size or location, and for moving the cursor within windows (e.g., 'scroll window'). Window commands are typically assigned to the four arrow keys and the "Return" key.

Menu Cursor Roam Commands Section

These commands allow users to move among menu choices when in menu-roam mode. These commands are usually assigned to the four arrow keys.

Menu Function Task Commands Section

These commands allow users to make menu choices with function keys.

Menu Prompt Line Terminators Section

These commands specify how the user can exit from the menu's prompt line. Users can go to the first or last roam-mode choice (if that mode is active), pick a choice, or go to the previous or top menu.

Conflicts in Command-to-Key Assignments

The Terminal Definer displays error or warning messages if there are conflicts in key assignments (i.e., two or more commands are assigned to the same key or keys). Note that an error or warning is issued if the complete command assigned to one key is the same as the beginning of a command assigned to another key. For example, the following command assignments are in conflict with each other:

```
exemac0:    004 013
exemac1:    004 013 010
```

An error message is issued if there are key-assignment conflicts between a command in a global section and any other section in the key-definition file. A warning message is issued if there are key-assignment conflicts between commands within a section or between local sections.

It is important to note that a key can have a different command assigned to it for different ALLY subsystems. However, command-to-key assignments in global sections take precedence over local key assignments.

You can assign the same command to more than one key (or set of keys) within a section. This allows the user to choose what keys to use when invoking the command. The following example shows the 'find' command assigned to two different keys.

```
find:033 117 122; PF3
find:033 133 061 176; Find key
```

A conflict occurs when a key (or given set of keys) has more than one command assigned to it. It is acceptable to have conflicts among sections and even within sections; however, key-assignment conflicts can cause some commands to become unavailable.

The rules for resolving key-assignment conflicts are:

- The commands in global sections of a key-definition file supercede those in any other section. Thus, if a command in the form/report section and a command in the global section are assigned to the same key (or keys), the global command assignment overrides the form/report command assignment.
- If you assign more than one command to the same key (or keys) within a section of the key-definition file, the first command assignment overrides any later assignments to that key within the section.
- If you assign a window command and another local command to the same key (or keys), the window command is invoked only when you type the key while in a window command function.

  For example, if 'window up' (window command) and 'up' (text editing command) share a key assignment, 'window up' is invoked only if the preceding command invoked a window operation ('scroll window', 'resize window', or 'move window').

Do not assign commands to ^Q (<Ctrl>Q) and ^S (<Ctrl>S), because the octal codes associated with ^Q (021) and ^S (023) are

part of the XON/XOFF protocol that ALLY uses to coordinate input and output with terminals. The Terminal Definer displays an error message and ignores a command assigned to a key that has 021 or 023 in its octal sequence. Also, because ALLY ignores the ASCII NUL character (octal 000), do not include the NUL character in key sequences.

Comments in a Key-Definition File

In a key-definition file, everything on a line following a semicolon (;) is treated as a comment and is ignored by the Terminal Definer. The comment symbol can be placed anywhere on the line.

The following example shows two placements of the comment symbol. First, all heading lines are commented out. On the last line, the Terminal Definer processes the mnemonic and octal code for the 'help' command but ignores the key assignment comment.

```
; Function  Octal Code      Key Assignment
;
;       GLOBAL COMMANDS
;
khelp:      001 010         ;^A^H
```

Figure 4-3. Comments in a Key-Definition File

The Ignore Command

You can assign the 'ignore' command to any key or sequence of keys that you want ALLY to ignore. Normally, when a key is pressed that is not assigned to an ALLY command, the key's octal sequence is displayed. However, when you press a key that is assigned the 'ignore' command, no action is taken and the display remains unaltered. This is useful when you want to leave a key without a command assignment. The ignore command can be assigned to keys in any section in the key-definition file. The following example shows how to assign the ignore command to the delete key.

```
ignore:    177              ; delete key
```

The ignore command's action depends on whether it is assigned in a global section or in a local section.

- A key assigned the ignore command in a global section is ignored throughout the application.
- A key assigned the ignore command in a local section is ignored only in that section.

Making Commands Unavailable to Users

You may not want a particular command to be available because it is not appropriate for your application. You can make a command unavailable to an application user with the key-definition file by:

- not assigning the command to a key
- assigning the ignore command to the key usually assigned to the command
- preceding the command mnemonic with a semicolon, which makes the key-to-command assignment a comment

In this example, the second key assignment for the 'expand window' command is commented out. (However, the user can still use the command through ALLY's Command Menus.)

```
expandwdw:  001 005              ; ^A^V
;expandwdw: 033 137 113 033 134 ; shift F3
```

User-Defined Key-Definition Files

You can define your own key-definition files by editing a copy of the key-definition file that is shipped with ALLY. Key-definition files that you define and store in a different directory are preserved when you install a new ALLY release.

Sample Key-Definition File

The command-to-key assignments in the following key-definition file are for a Unisys SVT-1220 terminal. The assignments were made for use by ALLY developers, so this key-definition file may differ from one used in a production environment:

- All commands in the list are defined—something few application users require.
- Many of the commands are assigned to obscure control character sequences (combinations of control character keys).
- Assumes that the SVT-1220's numeric keypad is in application mode rather than numeric mode (i.e., that keys send escape sequences, instead of numbers, when typed).

```
; COPYRIGHT: 1985, 1986, 1987, 1988 Foundation Computer Systems, Inc.
;
;    This software contains confidential and proprietary
;    information of Foundation Computer Systems, Inc.
;
;
;      ALLY Key definitions:  sample key-definition file
;
;      Standard key assignments for Unisys SVT-1220 terminals
;      UNIX Version Foundation Computer Systems
;
;     IMPORTANT NOTE:
;
;     The arrow keys and the home key can transmit more than one octal
;     code sequence.  Any ALLY command assigned to these keys must be
;     defined for all octal sequences that these keys can transmit.
;
;
; Function      Octal Code              Key Assignment
; ________      __________              ______________
;
;     GLOBAL FLAG DEFINITIONS AND SCROLL PERCENTAGE
;
overtype:       000                     ; overtype off
powertype:      000                     ; powertype off
drawmode:       000                     ; draw mode off
casesense:      001                     ; case sensitive
;                                         search on
findremain:     001                     ; don't move cursor
;                                         if find fails
findtoend:      000                     ; find end of string
;                                         off
clearfield:     001                     ; clear fields before
;                                         inserting
scrollprcnt:    012                     ; leave 10% of screen
;                                         on page scroll
;
;  The "@" marks the end of the first global section
@
;
;
;     GLOBAL COMMANDS
;
khelp:          001 010                 ; ^A^H
khelp:          033 133 062 070 176     ; Help key
abortaction:    003 001                 ; ^C^A
exitaction:     003 005                 ; ^C^E
aborttask:      003 003 001             ; ^C^C^A
exittask:       003 003 005             ; ^C^C^E
abortappl:      003 003 003 001         ; ^C^C^C^A
exitappl:       003 003 003 005         ; ^C^C^C^E
```

continued

```
shell:          033 117 120 033 117 167   ; PF1 7
shell:          003 010                   ; ^C^H
refresh:        001 032                   ; ^A^Z
setrptcnt:      033 117 121               ; PF2
prntvnum:       001 026                   ; ^A^V
prntscrn:       001 020                   ; ^A^P
prntscrn:       033 120                   ; ESC P
expandwdw:      001 005                   ; ^A^E
definewdw:      001 004                   ; ^A^D
compresswdw:    001 003                   ; ^A^C
movewdw:        001 015                   ; ^A^M
scrollwdw:      001 014                   ; ^A^L
explodewdw:     001 030                   ; ^A^X
resizewdw:      001 022                   ; ^A^R
toggletask:     001 061 015               ; ^A1^M
toggletask:     033 117 115               ; Enter
picktask:       024 020                   ; ^T^P
picktask:       033 117 120 033 117 115   ; PF1 ENTER
defmacro:       004 014                   ; ^D^L
mactofile:      004 027                   ; ^D^W
macfmfile:      004 022                   ; ^D^R
savemacros:     004 013                   ; ^D^K
loadmacros:     004 007                   ; ^D^G
exemac0:        005 060                   ; ^E0
exemac1:        005 061                   ; ^E1
exemac2:        005 062                   ; ^E2
exemac3:        005 063                   ; ^E3
exemac4:        005 064                   ; ^E4
exemac5:        005 065                   ; ^E5
exemac6:        005 066                   ; ^E6
exemac7:        005 067                   ; ^E7
exemac8:        005 070                   ; ^E8
exemac9:        005 071                   ; ^E9
exemacf:        005 006                   ; ^E^F
task:           001     003 013 061       ; DO key
task:           002     003 013 062       ; TASK menu
task:           003     003 013 063       ; MENU menu
task:           004     003 013 064       ; FORM/REPORT menu
task:           005     003 013 065       ; TEXT EDIT menu
task:           006     003 013 066       ; WINDOW menu
```

continued

```
task:           007     003 013 067     ; MACRO menu
task:           025     024 061         ; task1 key
task:           026     024 062         ; task2 key
task:           027     024 063         ; task3 key
task:           030     024 064         ; task4 key
task:           031     024 065         ; task5 key
task:           032     024 066         ; task6 key
task:           033     024 067         ; task7 key
task:           034     024 070         ; task8 key
task:           035     024 071         ; task9 key
@
;
;
;     TEXT EDITING COMMANDS
;
;
addnl:          015                     ; return
bdelete:        177                     ; DEL
bol:            033 117 163             ; 3 - ESC O s
bottom:         033 117 120 033 117 164 ; PF1 4
box:            002                     ; ^B
clrcasesens:    017 003 003             ; ^O^C^C
clrdrawmode:    017 004 003             ; ^O^D^C
clrovertype:    017 017 003             ; ^O^O^C
clrpwrtype:     017 020 003             ; ^O^C^C
cpfrombuf:      033 117 120 033 117 166 ; PF1 6
overlayblk:     017 014 003             ; ^O^L^C
overlayblk:     033 133 063 062 176     ; F18 key
cptobuf:        033 117 120 033 117 170 ; PF1 8
ctrlchar:       003 017                 ; ^C^O
delbol:         033 117 120 033 117 163 ; PF1 3
deleol:         033 117 120 033 117 162 ; PF1 2
delline:        033 117 123             ; PF4
deltomark:      033 117 166             ; 6
delword:        033 117 155             ; -
down:           033 117 102             ; down arrow
down:           033 133 102             ; down arrow
downpage:       033 117 164             ; 4
downpage:       033 133 066 176         ; Next Screen key
eol:            033 117 162             ; 2
fdelete:        033 117 154             ; ,
find:           033 117 122             ; PF3
find:           033 133 061 176         ; Find key
findanddel:     017 006 004             ; ^O^F^D
findanddel:     033 133 063 176         ; Remove                    continued
```

```
glblreplace:    033 117 120 033 117 171   ; PF1 9
hightomark:     017 010     010           ; ^O^H^H
hightomark:     033 133 063 063 176       ; F19 key
hightypeset:    017 010 005               ; ^O^H^E
home:           033 117 170               ; 8
ignore:         033 117 120 033 117 154   ; PF1 ,
ignore:         033 117 120 033 117 121   ; PF1 PF2
ignore:         033 133 062 176           ; Insert Here key
ignore:         033 133 064 176           ; Select key
ignore:         033 133 062 071 176       ; Do key
ignore:         033 133 061 067 176       ; F6 key
ignore:         033 133 061 070 176       ; F7 key
ignore:         033 133 061 071 176       ; F8 key
ignore:         033 133 062 060 176       ; F9 key
ignore:         033 133 062 061 176       ; F10 key
ignore:         033 133 062 063 176       ; F11 key
ignore:         033 133 062 064 176       ; F12 key
ignore:         033 133 062 065 176       ; F13 key
ignore:         033 133 062 066 176       ; F14 key
insertline:     033 117 120 033 117 160   ; PF1 0
jumptomark:     033 117 120 033 117 156   ; PF1 .
ldtomark:       017 010 014               ; ^O^H^L
ldtomark:       033 133 063 064 176       ; F20 key
left:           033 117 104               ; left arrow
left:           033 133 104               ; left arrow
left:           010                       ; left arrow
removeblk:      017 014 015               ; ^O^L^M
removeblk:      033 133 063 061 176       ; F17 key
mark:           033 117 156               ; .
nextline:       012                       ; linefeed
nextline:       033 117 160               ; 0
nextword:       033 117 161               ; 1
prevword:       033 117 120 033 117 161   ; PF1 1
readfile:       003 022                   ; ^C^R
redraw:         032                       ; ^Z
replace:        022                       ; ^R
replace:        033 117 171               ; 9
right:          033 117 103               ; right arrow
right:          033 133 103               ; right arrow
save:           033 117 167               ; 7
setcasesens:    017 003 005               ; ^O^C^E
setdrawmode:    017 004 005               ; ^O^D^E
setovertype:    017 017 005               ; ^O^O^E
setpwrtype:     017 020 005               ; ^O^C^E                    continued
```

```
togcasesens:    033 117 120 033 117 122   ; PF1 PF3
togcasesens:    017 003 024               ; ^O^C^T
togdrawmode:    017 004 024               ; ^O^D^T
togovertype:    017 017 024               ; ^O^O^T
togpwrtype:     017 020 024               ; ^O^C^T
top:            033 117 120 033 117 165   ; PF1 5
turtlehl:       017 024 010               ; ^O^T^H
turtleld:       017 024 014               ; ^O^T^L
turtleclear:    017 024 003               ; ^O^T^C
uldtomark:      017 010 025               ; ^O^H^U
uldturtle:      017 024 025               ; ^O^T^U
unbox:          025                       ; ^U
undelline:      033 117 120 033 117 123   ; PF1 PF4
undelword:      033 117 120 033 117 155   ; PF1
up:             033 117 101               ; up arrow
up:             033 133 101               ; up arrow
uppage:         033 117 165               ; 5
uppage:         033 133 065 176           ; Prev Screen
writefile:      003 027                   ; ^C^W
0
;
;
;     FORM/REPORT COMMANDS
;
;
insafter:       006 011 001               ; ^F^I^A
insbefore:      006 011 002               ; ^F^I^B
budmode:        006 002                   ; ^F^B
qbe:            006 025 002               ; ^F^U^B
qwhere:         006 025 027               ; ^F^U^W
query:          006 025 005               ; ^F^U^E
delrec:         006 004                   ; ^F^D
commit:         006 003                   ; ^F^C
rollback:       006 022                   ; ^F^R
dupdate:        006 024 004               ; ^F^T^D
pall:           006 024 001               ; ^F^T^A
ppage:          006 024 020               ; ^F^T^P
prest:          006 024 022               ; ^F^T^R
pickfield:      006 013 025               ; ^F^K^U
putfield:       006 013 004               ; ^F^K^D
pnext:          006 016 020               ; ^F^N^P
pprev:          006 020 020               ; ^F^P^P
phome:          006 006 020               ; ^F^F^P
plast:          006 014 020               ; ^F^L^P
```

continued

```
rghome:         006 006 007             ; ^F^F^G
rglast:         006 014 007             ; ^F^L^G
rgnext:         006 016 007             ; ^F^N^G
rgprev:         006 020 007             ; ^F^P^G
rhome:          006 006 022             ; ^F^F^R
rlast:          006 014 022             ; ^F^L^R
rnext:          006 016 022             ; ^F^N^R
rprev:          006 020 022             ; ^F^P^R
prhome:         006 006 001             ; ^F^F^A
prlast:         006 014 001             ; ^F^L^A
prnext:         006 016 001             ; ^F^N^A
prprev:         006 020 001             ; ^F^P^A
fhome:          006 006 006             ; ^F^F^F
flast:          006 014 006             ; ^F^L^F
fnext:          006 016 006             ; ^F^N^F
fnext:          015                     ; ^M
fnext:          033 117 102             ; down arrow
fnext:          033 133 102             ; down arrow
fprev:          006 020 006             ; ^F^P^F
fprev:          033 117 101             ; up arrow
fprev:          033 133 101             ; up arrow
fpickval:       006 026 020             ; ^F^V^P
flistval:       006 026 014             ; ^F^V^L
finsnext:       006 011 016             ; ^F^I^N
frfunction:     006 011 006             ; ^F^I^F
@
;
;
;       WINDOW COMMANDS
;
;
winup:          033 117 101             ; up arrow
winup:          033 133 101             ; up arrow
windown:        033 117 102             ; down arrow
windown:        033 133 102             ; down arrow
winright:       033 117 103             ; right arrow
winright:       033 133 103             ; right arrow
winleft:        033 117 104             ; left arrow
winleft:        033 133 104             ; left arrow
windone:        033 117 156             ; . - ESC O n
windone:        015                     ; carriage return
@
```

continued

```
;
;
;     MENU CURSOR ROAM COMMANDS
;
;
prevmch:        033 117 101             ; up arrow
prevmch:        033 133 101             ; up arrow
prevmch:        033 117 104             ; left arrow
prevmch:        033 133 104             ; left arrow
nextmch:        033 117 102             ; down arrow
nextmch:        033 133 102             ; down arrow
nextmch:        033 117 103             ; right arrow
nextmch:        033 133 103             ; right arrow
homemch:        010                     ; ^H
prompt:         020                     ; ^P
select:         015                     ; ^M
@
;
;
;     MENU FUNCTION TASK COMMANDS
;
;
menu:           001     003 015 061     ; ^C^M1
menu:           002     003 015 062     ; ^C^M2
menu:           003     003 015 063     ; ^C^M3
menu:           004     003 015 064     ; ^C^M4
@
;
;
;     MENU PROMPT LINE TERMINATORS
;
;

roamlast:       033 117 101             ; up arrow
roamlast:       033 133 101             ; up arrow
roamfirst:      033 117 102             ; down arrow
roamfirst:      033 133 102             ; down arrow
prevmenu:       025                     ; previous menu
topmenu:        024 015                 ; top menu
terminator:     015                     ;  ^M - CR
kmpprint:       017 015 020             ;  ^O^M^P
@
```

End of Chapter 4

Chapter 5
The Terminal Definition File

Introduction

This chapter describes the terminal definition file that the Terminal Definer uses when it builds an encoded Terminal Description file. The chapter also explains terminal capabilities, shows a sample terminal definition file that describes two terminals, and explains their entries.

The terminal definition file defines:

- capabilities for terminals on which ALLY can run
- the character sequences that invoke these capabilities

The terminal definition file is divided into sections, with each section describing a different terminal. The section for a given terminal defines the number of lines the terminal displays, the keys that move the cursor, the terminal's highlighting and line-draw capabilities, and its initialization sequences.

If there is not a section in the terminal definition file for your terminal type, you can write one. Look in the file named "termdef" to see if there is a definition section for your terminal type. (On UNIX systems, the "termdef" file is in the "/term/" subdirectory.)

Terminal Capability Entries

There are three types of terminal capability entries: Boolean, numeric, and string. The sequences that invoke a terminal's characteristics are described in the manufacturer's documentation for the terminal.

The Terminal Definer utility recognizes a standard notation for control characters in capability entries. The caret (^) signifies the character generated by pressing the control key (<Ctrl>) on a terminal along with the key specified (e.g., ^H means press the control key along with "H").

Boolean

Boolean entries specify whether a terminal has a given feature. ALLY has two Boolean capability codes:

bs means that ^H backspaces (moves the cursor one space to the left).

im means enter insert mode.

Numeric

Numeric entries specify counts and sizes for various terminal characteristics. The code is followed by a pound sign (#) and an integer number. For example, an entry of ":li#24:" specifies that the terminal displays 24 lines.

String

A string entry specifies that a capability is available, and provides the ASCII character sequence, or string, that invokes the capability. A string-capability code is followed by an equal sign (=) and the invocation sequence. For example, ":nd=^L:" specifies that ^L moves the cursor one space to the right.

Some terminals require a delay between executing a function and receiving the next character from ALLY. The length of this delay in milliseconds must be given as a decimal integer between the equal sign and the string. For example, ":cl=50\E[2J:" indicates that a "clear screen" takes 50 milliseconds on the terminal, and is invoked with the ASCII ESCAPE character (represented by "\E"), followed by these three characters: [2J. After sending this sequence, ALLY sends enough pad characters to provide the delay.

To list the character sequence for a string entry, use either the character itself or its octal code preceded with a back slash (\). For some commonly used ASCII characters that do not have a printable letter equivalent, use the following symbols:

\E	ASCII ESCAPE character, octal 033
\n	ASCII line feed (or new line), octal 012
\r	ASCII carriage return, octal 015
\t	ASCII horizontal tab, octal 011
\f	ASCII form feed, octal 014

Because the colon (:) is the capability entry delimiter, a colon that is part of a string entry must be represented with its octal code (\072). The Televideo 925 entry "h5=\041\EG\072:" illustrates the use of a colon within an entry. This entry defines the terminal sequence for invoking underline and blink highlighting.

Terminal Capability Codes

All terminal capability codes consist of two letters. Table 5-1 lists the code for each terminal capability, identifies its type, and describes the code. Codes for highlighting and line-draw capabilities are presented later in the chapter.

Table 5-1. ALLY Terminal Capability Codes

Code	Type	Description
al	string	Insert a blank line before the cursor line
bc	string	Move the cursor one space left (backspace), if not with ^H
bs	Boolean	Move the cursor one space left (backspace) with ^H
ce	string	Clear from the cursor position to the end of the line
cl	string	Clear the entire terminal display
cm	string	Move the cursor to a specific line and column
co	number	Number of columns displayed on a terminal line
dc	string	Delete one character at the cursor position
dl	string	Delete the line that the cursor is on
do	string	Move the cursor down one line
ei	string	End insert mode and enter overtype mode
ic	string	Insert a character at the cursor position, shifting text to the right
im	Boolean	Enter insert mode
is	string	Initialize the terminal for cursor movement and screen editing
ke	string	Exit special "keypad transmit" mode
ks	string	Enter "keypad transmit" mode, where the numeric keypad sends special command sequences instead of numbers
li	number	Number of lines displayed on a terminal
nd	string	Move the cursor to the right without overwriting text
pc	string	Pad character for delays (if not ASCII NUL \000)
rd	string	Downward scroll in a region of the display
rl	string	Left scroll in a region of the display
rr	string	Right scroll in a region of the display
ru	string	Upward scroll in a region of the display
tc	string	Cross-reference to another terminal that shares capabilities not listed in this terminal's description
te	string	Terminate cursor-motion mode
ti	string	Initialize cursor-motion mode
up	string	Move the cursor up one line

Sample Terminal Description File

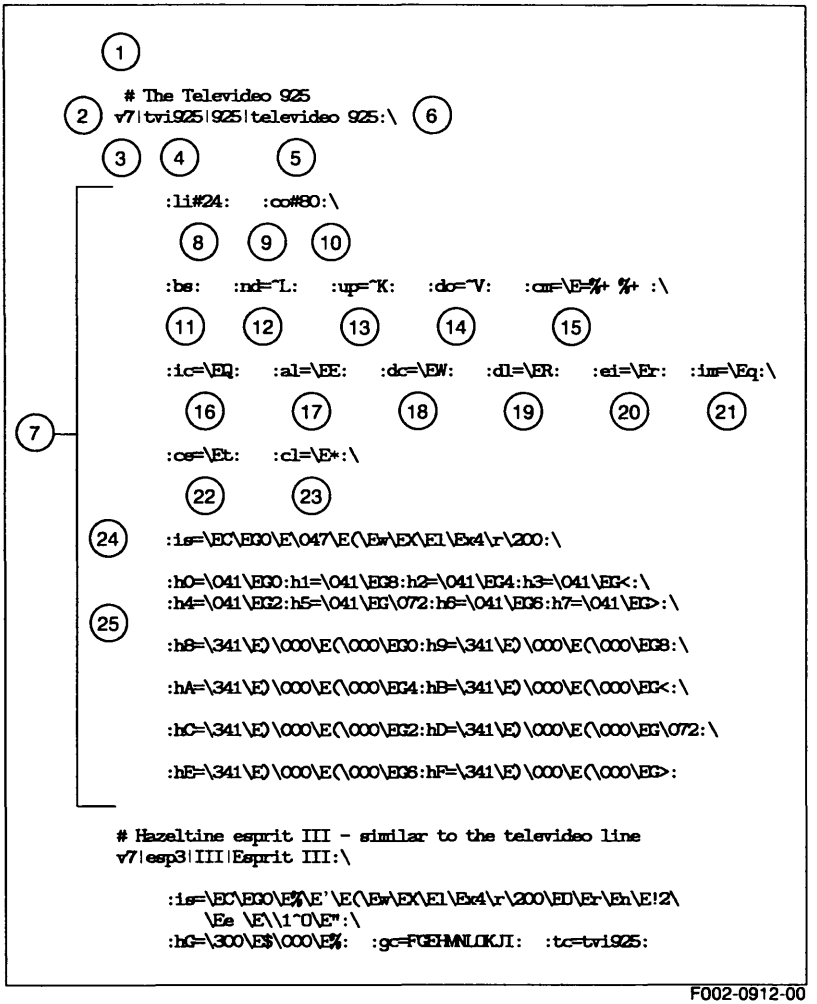
Terminal definition files are usually created by defining a terminal that has extensive capabilities. You can append sections for additional terminals to the file as needed by:

- defining only the capabilities that differ from the first terminal defined
- referencing the first terminal defined (with a "tc=" entry)

The first terminal defined in ALLY's terminal definition file is the Televideo 925. This terminal requires character positions on the screen to store a change in highlighting style. These positions, or spaces, cannot be used to display data. This is called *permanent highlighting* and is associated with a physical area on the screen.

The second terminal defined is the Hazeltine Esprit III because it also uses permanent highlighting. In addition, it has line-drawing capabilities. Figure 5-1 shows a terminal definition file with two sections that describe the Televideo 925 and the Hazeltine Esprit III.

The Televideo 925 section is described first. Note that the Televideo entries have been separated with blank lines for readability. However, the terminal definition file can have blank lines only between two sections; there cannot be blank lines within a section. Explanations of the capability codes defined in each section follow Figure 5-1.



F002-0912-00

Figure 5-1. Sample Printer Definition File

A Complete Terminal Description Section

The main components of the Televideo 925 section in the sample terminal definition file are:

- terminal names and editing capabilities
- initialization sequences
- highlighting capabilities

Explanations of the entries and the syntax rules are listed in the following pages. The numbers refer to Figure 5-1.

Terminal Names and Editing Capabilities

1) A comment can precede or follow each terminal description. The Terminal Definer considers any line that begins with the pound sign (#) in the terminal definition file to be a comment.

2) The first entry in a section contains at least three names for a terminal, separated by the vertical bar (|) and terminated by a colon (:).

3) The first name must be a two-character field that does not contain blanks. Usually this name is a letter followed by a digit, in this case, "v7." However, these two characters can be any that you choose to assign to a terminal. This name does not have to be unique within the terminal definition file.

4) The second name must be the mnemonic by which ALLY and the utilities reference the terminal. This should be a relatively short name consisting of letters and digits, in this case, "tvi925" for the Televideo 925 terminal.

5) The third and subsequent names can contain blanks. At least one should fully identify the terminal. In Figure 5-1, the terminal is also known as "925," and its complete name is "televideo 925."

6) Each line preceding the last line of a section must end with a back slash "\". A back slash signifies that the next line is a continuation of the section. The back slash must be the very last character on the line; do not leave a blank after it.

7) The remaining entries in a section describe the terminal capabilities. Each entry consists of a two-character code (e.g., nd) followed by a number or a character string and terminated by a colon (:). The order in which the capability codes are listed is arbitrary.

8) The Televideo's display is twenty-four lines long.

9) You can leave space between fields for readability.

10) The Televideo's display is eighty columns wide.

The Televideo can move the cursor:

11) One space to the left (backspace) with ^H.

12) One space to the right with ^L.

13) One line up with ^K.

14) One line down with ^V.

15) To a given line and column. (This entry is explained in the section, "Options for Cursor Movement Capabilities" at the end of this chapter.)

The next group of codes describes the sequences that invoke the insert/delete character and line functions:

16) Move the text at the cursor position to the right to insert a character.

17) Move the current line down one line to insert a blank line.

18) Delete the character at the cursor position.

19) Delete the line at the cursor position.

20) Enter overtype mode.

21) Enter insert mode.

The next codes describe the two strings that:

22) Clear the display from the cursor position to the end of the line.

23) Clear the whole display at one time.

Initialization Sequence

The initialization sequence contains the characters that ALLY sends to a terminal when execution begins. An initialization sequence sets up the terminal screen and keyboard for cursor movement and other screen editing functions. It should turn off unused features such as automatic margins, protected fields, and initial highlighting.

24) The initialization sequence for the Televideo 925:

- sets the terminal to conversational mode
- sets highlighting to normal display
- turns off protect and half-intensity modes
- turns off the automatic homing of the cursor when it gets to the end of the screen
- turns off monitor mode
- turns on the function keys for editing
- sets the terminal's "send" character to carriage return.

Highlighting Capabilities

25) The last entries describe the highlighting invocation sequences.

Highlighting Capabilities

ALLY uses four types of highlighting—underline, reverse video, blink, and altered intensity (higher or lower than normal). ALLY can use sixteen combinations of these four types. The codes for these are h0 through h9, and hA to hF (A through F are the hexadecimal digits for decimal 10 through 15). Table 5-2 lists the ALLY highlighting codes by the type of highlighting they produce.

Table 5-2. ALLY Highlighting Codes

		Highlight Codes			
Code	**Type**	**Underline**	**Reverse Video**	**Blink**	**Altered Intensity**
h0	string	-	-	-	-
h1	string	x	-	-	-
h2	string	-	x	-	-
h3	string	x	x	-	-
h4	string	-	-	x	-
h5	string	x	-	x	-
h6	string	-	x	x	-
h7	string	x	x	x	-
h8	string	-	-	-	x
h9	string	x	-	-	x
hA	string	-	x	-	x
hB	string	x	x	-	x
hC	string	-	-	x	x
hD	string	x	-	x	x
hE	string	-	x	x	x
hF	string	x	x	x	x

For example, the code "h0" sends the terminal sequence that turns off all highlighting, and "hA" sends the sequence that produces reverse video with altered intensity. Each ALLY highlighting code turns on the highlighting styles it represents and turns off all the others.

Permanent Highlighting

The video attribute codes for the Televideo 925 and Hazeltine Esprit III terminals require a space on the display screen. That is, one character space on the screen must be reserved for the code that turns on highlighting and another space must be reserved for the code that turns off highlighting. This is called permanent highlighting because it is associated with a physical area on the screen. Because permanent highlighting can require up to three spaces before and after the highlighted area to mark it, these spaces cannot display characters.

Permanent highlighting marks the beginning and end of an area on the terminal screen to be permanently highlighted. If new text overwrites text in a permanent area, the new text is highlighted.

Transient Highlighting

Another type of highlighting, called *transient*, is associated with the highlighted text and does not require extra display space. Text sent to the terminal between the time that transient highlighting is turned on and the time that transient highlighting is turned off is highlighted. If the text changes, the highlighting disappears. Because transient highlighting does not require space for marks before and after the highlighted text, the entire area can be used to display characters.

Highlighting Sequences

The first part of a highlighting entry, called a *flag byte*, comes before the invocation sequence. The flag byte is not sent to the terminal, but instead, tells ALLY how a terminal handles highlighting. Some terminals must have multiple codes to invoke a highlighting style. For example, the Televideo 925 must send a code for reverse video and a code for altered intensity to highlight an area with reverse video at altered intensity. The SVT-1220, however, needs only one code to highlight with reverse video at altered intensity.

The flag byte tells ALLY whether a highlighting sequence consists of a single code or multiple codes. The flag byte for multiple highlighting codes tells ALLY which codes are for permanent highlighting and which are for transient.

To calculate the value of the flag byte, sum the octal codes (from the list below) in the highlighting sequence, plus the number of spaces (0-3) needed to mark the beginning and end of a permanent highlight area. The octal code values that comprise a highlighting sequence are:

\200 Transient-On
\100 Transient-Off
\040 Permanent

The value of the flag byte reflects the number of codes (one to three) the highlighting sequence contains. For example, a highlighting sequence containing only a transient-on code has a flag byte value of octal 200. The flag byte for a sequence containing a transient-on code, a transient-off code, and a permanent-on code requiring one space to set, has an octal value of 200+100+040+1, or 341. Note that there is only one code for permanent highlighting (\040). Permanent highlighting remains on until the code to turn off highlighting (h0) is sent.

If the highlighting sequence consists of more than one code, they *must* be in the order listed above. Each code that is followed by another code must end with \000. For example, in the entry "h8=\341\E)\000\E(\000\EG0:" the first \000 tells ALLY that this is the end of the transient-on code and that the beginning of the transient-off code follows. If a sequence string must contain an ASCII NUL (\000), encode that NUL as \200 to avoid confusion.

The following expansions of two of the sixteen highlighting sequences for the Televideo 925 terminal illustrate this process. The sequence "h2=\041\EG4:" turns on reverse video. The flag byte, \041, tells ALLY that the sequence contains a permanent-on code requiring one space to set. To highlight text in reverse video, ALLY sends "\EG4" to the terminal, then sends the text, and then sends "\EG0" to turn off highlighting. (The \EG0 sequence is defined in the Televideo's terminal definition file for the ALLY code that turns off all highlighting, "h0".)

The sequence "h8=\341\E)\000\E(\000\EG0:" turns on altered intensity. The value of the flag byte, \341, tells ALLY that the sequence contains a transient-on part "\E)", a transient-off part "\E(", and a permanent-on part "\EG0" that requires one space to mark. To highlight text in altered intensity, ALLY sends to the terminal "\EG0\E)", then the text, then "E(\EG0".

Table 5-3 separates the Televideo highlighting sequences to make them more readable.

Table 5-3. Televideo Highlighting Sequences

Terminal Definition Code	Flag Byte	Sequence That Invokes		
		Transient-On	Transient-Off	Permanent-On
:h0	\041			\EG0
:h1	\041			\EG8
:h2	\041			\EG4
:h3	\041			\EG<
:h4	\041			\EG2
:h5	\041			\EG\072*
:h6	\041			\EG6
:h7	\041			\EG>
:h8	\341	\E)°	\E(°	\EG0
:h9	\341	\E)°	\E(°	\EG8
:hA	\341	\E)°	\E(°	\EG4
:hB	\341	\E)°	\E(°	\EG<
:hC	\341	\E)°	\E(°	\EG2
:hD	\341	\E)°	\E(°	\EG\072*
:hE	\341	\E)°	\E(°	\EG6
:hF	\341	\E)°	\E(°	\EG>

* 072 is the octal representation of a colon

° In the terminal definition entry, this part of the sequence is followed by the separator \000.

Line-Draw Capability Section

In addition to the standard Televideo 925 terminal features, the Hazeltine Esprit III terminal can produce line-draw characters for making boxes with corners on the display. Table 5-4 describes the line-draw codes available.

Table 5-4. Line-Draw Codes

Line-Draw Codes		
Code	**Type**	**Description**
gc	string	List of letters that specify line-draw characters
hG	string	Initialize line-draw characters and strings (used with the characters in gc entry)

Figure 5-2 shows the section of the sample terminal definition file that describes the Esprit. This section lists only an initialization sequence and the line-draw sequences, and references the Televideo 925 for the remainder of its characteristics. We have inserted blank lines for readability, but a terminal definition file can contain blank lines only between sections.

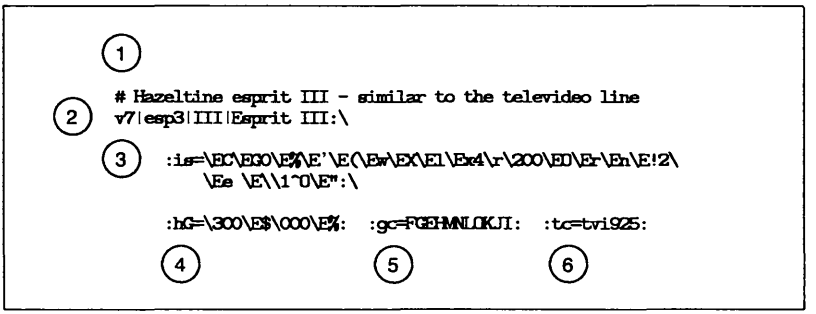


Figure 5-2. Hazeltine Esprit III Terminal Section

1) The description begins with a comment about the terminal.

2) This entry lists the names by which this terminal is known.

3) The Esprit III requires a separate initialization sequence to ensure that line-draw mode is off when ALLY initializes the terminal.

4) The "hG" entry has a flag byte that tells ALLY that the sequence contains a transient-on code and a transient-off code for turning line-draw mode on and off. To draw boxes and lines on the Esprit III, ALLY must send it "\E$", followed by the letter(s) listed in the "gc" entry to draw the lines, followed by "\E%" to turn off line drawing.

Line-Draw Capability

5) The "gc" entry lists letters for the eleven line-draw characters that the Esprit III can produce. The letters that represent the line-draw characters must be listed in this order:

    1 upper-left corner
    2 upper-right corner
    3 lower-left corner
    4 lower-right corner
    5 left T-bar (intersection of horizontal line with the middle of a vertical line to its left)
    6 upper T-bar
    7 right T-bar
    8 lower T-bar
    9 horizontal line
    10 vertical line
    11 a cross (vertical line and horizontal line both intersecting in the middle)

6) The last entry, "tc=tvi925," tells ALLY to obtain the remaining capabilities for the Esprit III from the "tvi925" entry. The "tc" entry, if included in a terminal's description, must come last in the section.

The "tc" entry references the characteristics of another terminal in the terminal definition file. The terminal being described shares all of the capabilities of a terminal referenced by "tc" except those itemized in the terminal's section. ALLY checks the terminal named in the "tc" entry for each capability not listed in the terminal's description.

A terminal definition file usually contains a full description for the terminal with the most capabilities. Sections for other terminals that share many of these capabilities list only capabilities that are different, and end with a "tc" entry.

Options for Cursor Movement Capability

This section describes the options for the "move cursor to line X, column Y" terminal definition entry, "cm". The cursor movement options are listed and described in Table 5-5. These options are specified after the sequence that signals cursor motion in a "cm" sequence.

A "move cursor" sequence directs a terminal to move its cursor to a specified line and column. Terminal manufacturers use different ways to encode this, so the "cm" terminal definition entry must be capable of describing many different formats. The manufacturer's documentation for the terminal specifies its requirements.

Some terminals require that the "move cursor" sequence give the line first, then the column, while other terminals require the opposite. Some terminals number lines and columns beginning with zero, while others begin with one. Some require complex encodings of the line and column numbers. These are all specified with options in the "cm" sequence.

The "cm" sequence options are marked with percent signs, "%," followed by one or more letters that specify the option. The line and column numbers are substituted in the "cm" string in place of the option when the string is sent to the terminal.

Table 5-5. Cursor Movement Options

Option	Description
%0	The order of the values to be sent out for region scrolling.
%r	Reverse the line and column output order—send column first, then line. By default, ALLY substitutes the line number followed by the column number.
%i	Increment the line and column numbers. By default, ALLY starts lines and columns at 0; with this option, ALLY starts lines and columns at 1.
%>ab	If either the line or the column number is greater than the value of the number in the 'a' position (expressed in octal), then add to that number the value of the number in the 'b' position (also expressed in octal). In the example "%>\010\050," a line value of 5 is left as is, and a column value of 12 is converted to 52 (octal 050 is decimal 40, plus 12 is decimal 52).
%n	Exclusive-Or the line and column numbers with octal 140.
%B	Convert the line and column numbers to BCD (Binary Coded Digits) before sending them.
%D	For each line and column number: divide by 16, multiply the remainder by 2, and subtract the result from the line or column number. This is required by the Delta Data terminal.
%%	Send a single percent sign.

The remaining options must be placed in the sequence *after* any of the preceding options.

Option	Description
%d	Send the line or column number as an integer. For example, ALLY sends column 25 as the two characters "25."
%2	Same as %d, but send the number as two digits. For example, ALLY sends 2 as "02" and 79 as "79."
%3	Same as %d, but send the number as three digits. For example, ALLY sends 6 as "006" and 125 as "125."
%+n	Add the octal value of the number in the 'n' position to the number and send the result as one character. For example, "%+A" sends 5 as ASCII "F" (5 + 101, which is the value of A, = 106, which is the value of F).
%.	Send the number as the ASCII character equivalent of its octal value. For example, ALLY sends column 39 as a single quote (') since ASCII 39 equals octal 047, and octal 047 is a single quote.

The sequence to move the cursor to line X and column Y on the Televideo 925 is "cm=\E=%+ %+ ". It begins with the two ASCII characters, ESCAPE and equal sign (\E=), followed by the ASCII character whose octal value is the line number added to the octal value of an ASCII blank " ", followed by the ASCII character whose octal value is the column number added to an ASCII blank. For example, ALLY sends "\E= &" to the terminal to move the cursor to row 0, column 6.

User-Defined Terminal Definition Files

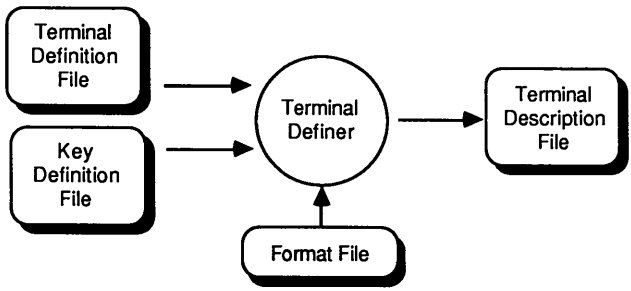
If you create separate terminal definition files and store them in a different directory, they are preserved when you install a new ALLY release. However, if you define a terminal by appending its description to the terminal definition file that is shipped with ALLY, it is lost when you install a new ALLY release.

End of Chapter 5

Chapter 6
The Terminal Definer Utility

Introduction

The Terminal Definer builds the terminal description file that provides ALLY's portability from terminal to terminal. ALLY, and applications built with it, execute on most RS232 ASCII terminals that support a "position cursor to x,y" function. The terminal description file contains information about the highlighting abilities, line drawing abilities, and optimizations for each terminal an application uses. Figure 6-1 shows the basic operation of the Terminal Definer.



F002-0556-00

Figure 6-1. The Terminal Definer

Terminal Definer Input

The Terminal Definer utility requires input from:

- the Format File
- the terminal definition file
- the key-definition file

The sections of the Format File that the Terminal Definer uses are described in Chapter 3. The next two sections discuss the terminal-definition and key-definition files.

The Terminal Definition File

The ALLY terminal definition file provides the Terminal Definer with information about the capabilities of the terminals that an application uses. The terminal definition file supplied with ALLY contains descriptions for several common terminals. If a terminal that your application uses is not currently listed in the terminal definition file, you can build a terminal definition file for it. Chapter 5 describes the terminal definition file.

The Key-Definition File

Any ALLY command that can be invoked by a key can be assigned to any key or key sequence. (Throughout this manual, "key sequence" can be substituted for "key.") These assignments are made in a terminal's key-definition file. The key-definition file assigns commands to keys by specifying the key's octal code or code sequence next to the command mnemonic. Chapter 4 describes the key-definition file.

Terminal Definer Options

The Terminal Definer displays error or warning messages if there are conflicts in the commands assigned to keys (i.e., two or more commands assigned to the same key or keys). The Terminal Definer issues an error message if there are command-assignment conflicts between a global section key and a key in any other section in a key-definition file. The Terminal Definer issues a warning message if there are key-assignment conflicts between commands within a section or between commands in local sections.

It is acceptable to have conflicts among sections and even within sections; however, key-sequence conflicts can cause some commands to become unavailable to users. The rules for resolving key-assignment conflicts are described in Chapter 4.

Table 6-1 shows the Terminal Definer options for checking conflicts in key assignments in the key-definition file.

Table 6-1. Terminal Definer Options

Option	Action
Every key with global keys	Compare every key in the key-definition file for conflicts with the keys assigned in the two global keys sections.
Each key with every key	Compare each key in the key-definition file with every other key in that file for an assignment conflict.
Each key with the same section	Compare each key in the key-definition file with other keys in the same section for an assignment conflict.

Option	Action
No conflict checking	Do not compare keys in the key-definition file for assignment conflicts.

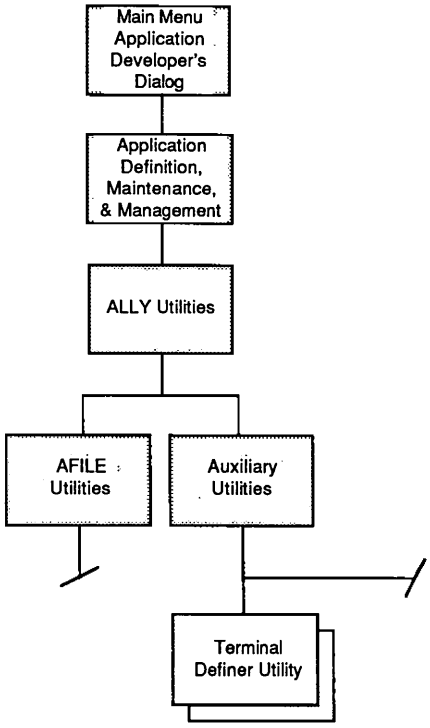
Terminal Definer Output

The output produced by the Terminal Definer is an encoded file that contains all of the information needed for an ALLY application to run on a specific terminal. This output file is optimized for ALLY execution time.

The name of this encoded file is the first argument you specify when you invoke ALLY. The distributor, application developer, or end-user can run the same application on different terminals by changing this single argument in the ALLY command. Or, a system manager can build these arguments into operating-system command files for each terminal in the system.

Invoking the Terminal Definer from the Dialog

Figure 6-2 shows the location of the Dialog forms you use to invoke the Terminal Definer.



F002-0583-01

Figure 6-2. Dialog Path to the Terminal Definer

Figure 6-3 shows the Dialog form that you use to run the Terminal Definer.

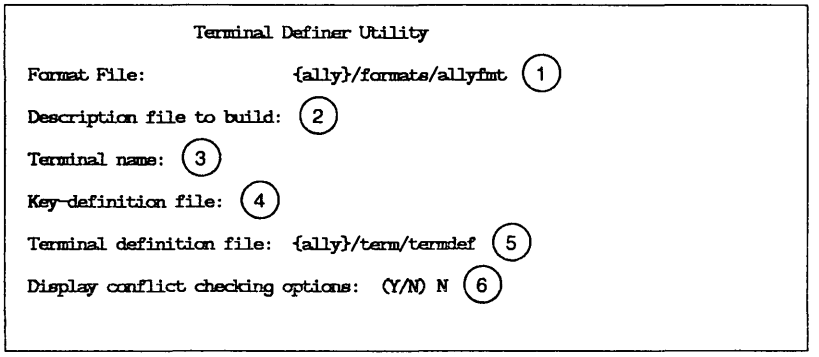


Figure 6-3. Invoking the Terminal Definer

1) The name of the current Format File is filled in. Type <Return> to use this file, or, edit or delete the displayed name to use a different Format File.

Enter the following information in the remaining fields:

2) Specify a name for the file that is to contain the terminal's character mapping information. The file name must be compatible with your operating system's naming conventions.

3) Type the ALLY mnemonic from the terminal definition file that identifies the terminal for which you are building this description file.

4) Type the name of the file that specifies the key-to-command assignments for the terminal named above.

5) The name of the file that lists terminal capabilities when ALLY is shipped is filled in. If you want to use a different terminal definition file, type in the name of the one you want.

6) Type <Return> to accept the default level of conflict checking—that keys are not checked for assignment conflicts. The cursor moves to the confirmation field.

   Type "Y<Return>" to display the subform (Figure 6-4) that lists the conflict checking options.

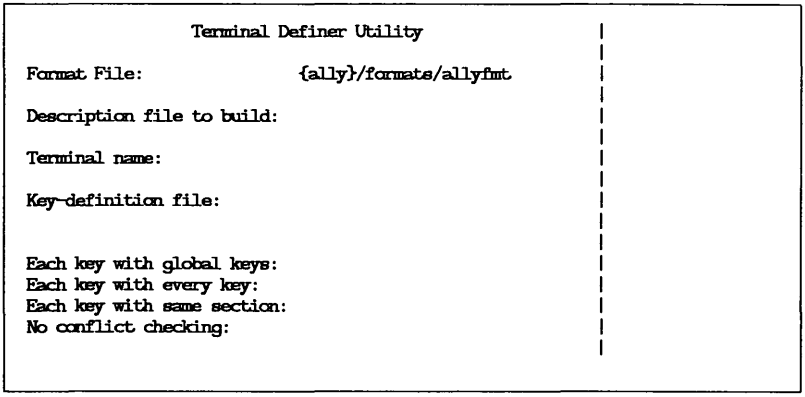


Figure 6-4. Terminal Definer Conflict Checking Options

Type "X<Return>" or "Y<Return>" in the field of a level of conflict checking that you want to select.

After you enter the required information and use the 'exit action' command, the Terminal Definer executes. The cursor moves to the lower-left corner of the form while the Terminal Definer runs. After the processing is completed, the cursor moves to the menu titled *Auxiliary Utilities*.

Invoking the Command File for the Terminal Definer

The command file for this utility automatically uses the current Format File and terminal definition file.

The Terminal Definer command file and its arguments are:

newterm *[terminal name] [output file] [keydef file] [option]*

This command requires only the first three arguments. If you choose to specify only three arguments, the Terminal Definer does not check for conflicts in key assignments.

You must specify these arguments:

terminal name — The ALLY mnemonic from the terminal definition file that identifies the terminal for which the terminal description file is being built.

output file — The terminal description file produced that contains the key-mapping information for the terminal specified by "terminal name."

keydef file — The name of the file that contains the command-to-key assignments for your keyboard.

options — One of the options that specifies the level of conflict checking for key assignments. The Terminal Definer options for conflict checking level are listed below.

- a — compare each key with every other key for a conflict in assignment
- g — compare each key with the keys assigned in the two global sections for an assignment conflict
- n — do not check for conflicts in key assignments
- o — compare each key with other keys in the same set for an assignment conflict.

Invocation Example

```
newterm tvi925 descript tvi925.kdf a
```

This example invokes the command file for the Terminal Definer to build a terminal description file for a Televideo 925 terminal. The description file produced is named "descript." The ALLY mnemonic that identifies the Televideo 925 terminal in the terminal definition file is "tvi925." Its key-definition file is named "tvi925.kdf". The "a" option checks all keys for command assignment conflicts.

Terminal Definer Error Messages

The Terminal Definer produces an error message if it finds:

- no entry in the terminal definition file for the named terminal
- an invalid octal number in the key-definition file (such as 089)
- an illegal character (^Q or ^S) in the key-definition file
- a line in the key-definition file that is not in the correct format
- a conflict between two command-to-key assignments
- an incorrect number of arguments
- an inaccessible terminal definition file

End of Chapter 6

Chapter 7
The Printer Definition File

Introduction

ALLY applications can use any standard ASCII printer. The printer definition file provides the Printer Definer utility with information about the features of the printers that ALLY applications use. These features include boldfacing and underscoring.

The Printer Definer uses the printer definition file to create an encoded file that identifies a printer to ALLY. ALLY provides a printer definition file named "printdef" that describes several printers. (On UNIX systems, this file is in the"/printers/" subdirectory.) Look at this file to see if there is a printer description for your printer.

If a printer your application uses is not currently in the printer definition file, you can either add its description to the "printdef" file or create a separate printer definition file in a different directory. The latter method preserves printer descriptions that you create when you install a new ALLY release. The printer definition file is very similar to the terminal definition file. Before you try to create a printer description for a printer definition file, you should read Chapter 5, which explains how to create terminal descriptions for a terminal definition file.

ALLY provides commands that print:

- the current display image
- the current page of a form/report
- all remaining pages of a form/report
- all pages of a form/report

All images, whether a single screen or an entire report, are translated from ALLY's standard internal format to a format suitable for the specified printer. The printer definition file gives ALLY information on how to translate display images to print files. When a print command is issued, ALLY routes the request to the appropriate printer and transforms the material to a printable form by translating:

- formatting characters (such as newlines and page-breaks) from a display format to a format for printing
- highlighting (such as reverse video) to a form that can be printed

ALLY tries to optimize the print format for the printer's requirements.

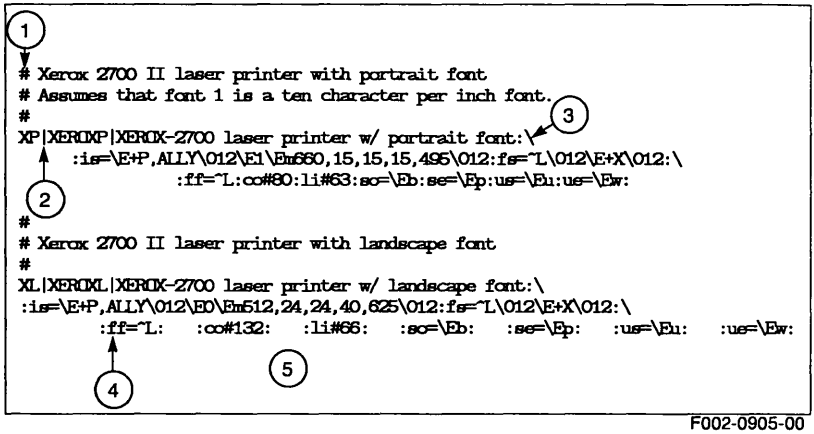
Syntax of the Printer Definition File

The printer definition file encodes printer features in the same way that the terminal definition file encodes terminal features. A printer description in the printer definition file starts with a list of names, or mnemonics, by which a printer is known (e.g., XEROXP for the Xerox 2700 II Laser Printer with portrait font). The remainder of the printer description enumerates capabilities and the characters necessary to invoke them.

Figure 7-1 shows the format of printer definition file entries for the Xerox 2700 II Laser Printer. There are two descriptions for the Xerox 2700:

XEROXP Portrait font that prints pages that are 63 lines long, 80 columns wide

XEROXL Landscape font that prints pages rotated sideways, 63 lines by 132 columns



F002-0905-00

Figure 7-1. Sample Printer Definition File

1) Comment lines begin with a pound sign (#).

2) A description section begins with the printer's mnemonics, separated by "|".

3) Continuation lines within each entry end with "\".

4) Each printer capability is represented with a two-letter code. A capability can be Boolean, numeric, or string (see Table 7-1). The manufacturer's documentation for the printer specifies the sequences that invoke each capability.

5) Fields are delimited with a colon (:). Fields can be separated with spaces for readability.

Printer Features

Printers vary in the way they handle margins, pagination, and highlighting.

Page Margins

Characters exceeding the printer's maximum line length are printed on the next line, starting in column one. Note that this can affect pagination.

Pagination

To move the printer to the next page, a form feed (also called a page eject) normally follows each page that is printed. When a page exceeds the printer page length, the extra lines are printed on the top of the next physical page. A form feed follows the last line on the page.

The form-feed string causes the printer to go to a new page. If this string is set to null (:ff=:), ALLY assumes the printer does not provide the form-feed function. In this case, ALLY outputs blank lines to reach a new page. It is important to note that there is a difference between omitting the form-feed string and setting it to null. If you omit the form-feed string, the default value is ^L (see Table 7-1).

Aside from how the printer advances to the next form, you can specify a string to pad output with leading and trailing blank pages before and after each print job. Even if these strings are not specified, ALLY automatically advances to the next form after the last page so that the next print job does not start on the same page as the previous job.

Highlighting

ALLY supports sixteen highlighting styles for a display. However, standard printers are capable of only a few styles of text emphasis. Therefore, ALLY supports only two terminal display highlighting modes for printed output: reverse video and underline, and the combination of these two.

Reverse video is translated to boldface, and underline on the display is translated to underlining on the printed page. All other display highlighting is ignored.

Highlighting is accomplished in the best way available on each printer. For efficiency and visual effect, highlighting is usually done by issuing the appropriate start and stop highlighting commands. Where the string to be highlighted spans multiple print lines, ALLY automatically turns off highlighting at the end of each line and turns on highlighting again at the beginning of the next line.

These highlighting commands are not available on some printers. In this case, ALLY uses a carriage return string to return to the start of the line and then overprints the highlighted string. Highlighting is not possible on printers that do not have either special commands to turn highlighting on and off or an explicit carriage return string. In addition, the last character of each line cannot be highlighted when the printer supports automatic margins.

Printer Capability Codes

Table 7-1 lists each printer capability, identifies its type, and describes it.

Table 7-1. ALLY Printer Capability Codes

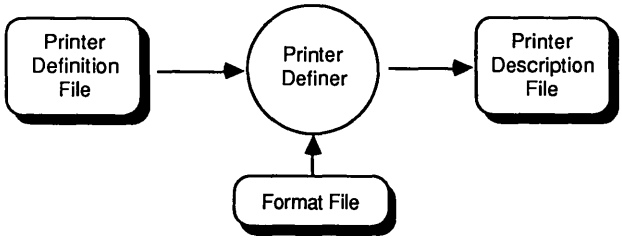
Name	Type	Description
am	Boolean	Automatic margins
ap	Boolean	Automatic end of page
bc	string	Backspace if not ^H
bs	Boolean	Backspace with ^H
co	number	Number of columns in a line
cr	string	Carriage return (default ^M)
ff	string	Form feed (default ^L)
fs	number	If printer is a tty, set flag bits (default 0)
is	string	Printer initialization string
li	number	Number of lines on a page
nl	string	Newline character (default 0)
os	Boolean	Printer overstrikes
se	string	End standout mode
so	string	Begin standout mode
ue	string	End underscore mode
us	string	Start underscore mode

End of Chapter 7

Chapter 8
The Printer Definer Utility

Introduction

The Printer Definer, shown in Figure 8-1, requires input from the Format File and the printer definition file. The printer definition file provides the Printer Definer with information about the capabilities of the printers that your application uses. See Chapter 7 for information about the printer definition file.

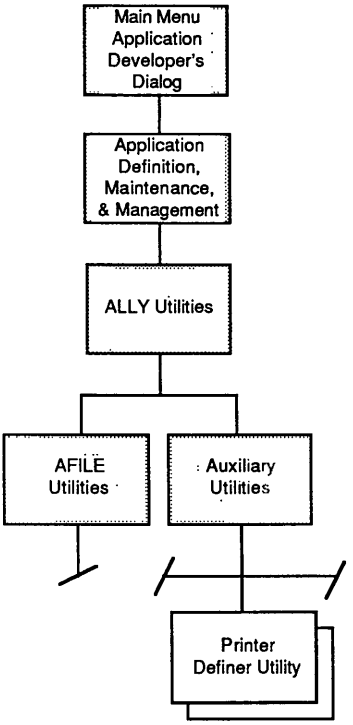


F002-0557-00

Figure 8-1. The Printer Definer

Invoking the Printer Definer from the Dialog

Figure 8-2 shows the location of the Dialog forms you use to invoke the Printer Definer.



F002-0818-00

Figure 8-2. Dialog Path to the Printer Definer

Figure 8-3 shows the Dialog form that you use to invoke the Printer Definer.

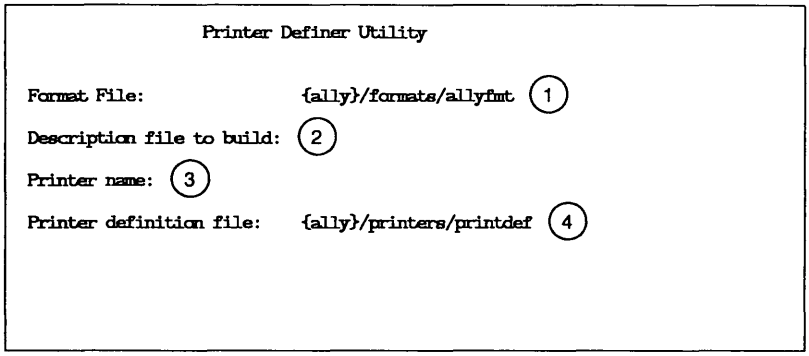


Figure 8-3. Invoking the Printer Definer

1) The name of the current Format File is filled in. Type <Return> to use this file, or, edit or delete the displayed name to use a different Format File.

Enter the following information in the remaining fields:

2) Specify a name for the printer description file you want ALLY to build. This name must be compatible with your operating system's naming conventions.

3) Type the ALLY mnemonic from the printer definition file that identifies the printer for which the printer description file is being built.

4) The name of the current printer definition file is filled in. Type <Return> to use this file. To use a different file, type in the name of the file that lists the capabilities of the printer for which you are building the description file.

After you enter the required information and use the 'exit action' command, the Printer Definer executes. The cursor moves to the lower-left corner of the form while the Printer Definer runs. After processing is completed, the cursor moves to the menu titled *Auxiliary Utilities*.

Invoking the Command File for the Printer Definer

The command file for this utility automatically uses the current Format File and printer definition file.

The Printer Definer command file and its arguments are:

newprint *[printer name] [output file]*

You must specify for the arguments:

printer name — The ALLY mnemonic from the printer definition file that identifies the printer for which the printer description file is being built.

output file — The printer description file to be produced for the printer specified by "printer name."

Printer Definer Error Messages

The Printer Definer produces an error message if it finds:

- an output file that cannot be opened
- a printer without an entry in the printer definition file

End of Chapter 8

Chapter 9
Managing Printer Output

Introduction

Print commands for ALLY applications can be issued either:

- automatically by any form/report, or
- explicitly by the user.

ALLY builds the print file and sends it to the printer with the operating system's standard facilities. You tell ALLY how you want to handle printing for an AFILE by providing information about the following:

- The *printer description file* is the encoded file produced by the Printer Definer utility that tells ALLY how to interpret screen images for the printer that prints your application's reports.

  If you have more than one printer, you can choose which printer to use for printing your application's forms/reports by specifying a different printer description file. You do this by modifying the application with the Dialog.

- The *printer output file* contains the interpreted screen images of your application's reports. This is the default destination for form/report output. Each print request to this file writes over the previous file contents. The file is not printed until you invoke your operating system's print command.

  You specify a printer output file for your application AFILE with the Dialog. If you do not specify a printer output file, the print image is written to a temporary file that is deleted when the file is printed.

- The *spooling information* names the destination for ALLY print requests, and can be a physical device or a printer queue. It can also be the name of a file if you have assigned "copy" as the value of the "allyprinter" environment variable. (See the section, "Overriding Printer Specifications," for information about the "allyprinter" variable.)

  You specify spooling information for your AFILE with the Dialog and set environment variables with operating system commands.

Spooling

Translated display images are written to a print file and sent to the operating system's print spooler. Images can also be spooled to a standard text file that can be edited.

If spooling devices are available, spooling is the default. ALLY provides a program that simulates a spooler for UNIX operating systems without print spoolers. The "spoolcmd" file is ALLY's interface to UNIX printers. There are two "spoolcmd" files located in the "/printers/" subdirectory:

spoolcmd	for UNIX systems with spooling devices.
spoolcmd.noq	for UNIX systems without spooling devices.

Follow the steps below if you are not using spooling devices:

1) Make a backup copy of the "spoolcmd" file by copying the file to a file with a different name. For example,

```
cd printers
cp spoolcmd spoolcmd.que
```

2) Write the "spoolcmd.noq" file over the "spoolcmd" file with the 'copy' command. For example,

```
cp spoolcmd.noq spoolcmd
```

ALLY requires a passive spooling device, that is, one that does not interpret the characters sent to it. If your spooling device is not a passive service, you need to change the printer definition file specifications for special and control characters.

Overriding Printer Specifications

You can append the contents of the printer output file to a spooling file and override an application's global printer specification by defining an environment variable named "allyprinter." You assign a value to the "allyprinter" variable with an operating system command before running an application.

The "allyprinter" environment variable can be set to one of three values:

none	Sends interpreted screen images of your application's reports to the printer output file. Use "none" when you do not want your output to be routed to a printer. Each print request to this file writes over the previous file contents.
copy	Sends interpreted screen images of your application's reports to the printer output file, and then appends the file's contents to the spooling file. The file is not printed until you invoke your operating system's print command. The printer subsystem continues to append images from the printer file to the spooling file until you delete the spooling file.

queue_name Sends interpreted screen images of your application's reports to the printer output file and then routes the file to the specified printer queue.

Specify a valid spooling device for *queue_name*. A queue name specified for the value of "allyprinter" overrides the default spooling device that is displayed in the Dialog's and AMU's *Global Printer Information* form.

Neither "none" nor "copy" is case sensitive, though "allyprinter" is. The "allyprinter" variable overrides all other printer routing information for the application. If the "allyprinter" variable is not defined, output is sent to the printer output file and routed to the printer or device that is defined for the application. See Chapter 1 for more information about environment variables.

End of Chapter 9

Appendix A
UNIX ALLY Directory Structure

This appendix lists the subdirectories and files contained in the top-level release 2.00 ALLY directory named "ally2_00."

Subdirectory	Contains	
	Subdirectory	**File**
	* Development systems only	
afiles	allyedit	allyedit.a
	amu	amu.a, amu.e, and amu.h
	commen	commen.h
	dialog*	dialog.a*, dialog.e*, and dialog.h*
	errors	errors.e
	storybook	start.a, HOURS.A, salary.fx, EMPLOYEE.fx, HOURS.fx
allyexe		allydevx*
		allyrunx
		amigratx*
		amux
		compactx*
		dmigratx*
		mergex*
		mmigratx
		newmsgx*
		newprntx
		newtermx
		scriptx*
bin		acompact*
		ally
		allyedit
		amerge*
		amigrate*
		amu
		ascript*
		dialog*
		dmigrate*

Subdirectory	Contains	
	Subdirectory	**File**
	* Development systems only	
		mmigrate
		newmsg*
		newprint
		newterm
formats		allyfmt
objects*		usercode.o*
printers		DFLT
		LP05
		printdef
		spoolcmd
		spoolcmd.noq
src*	include*	usercode.in*
term		svt100c
		svt100c.kdf
		svt1210
		svt1210.kdf
		svt1210a
		svt1210a.kdf
		svt1210c
		svt1210c.kdf
		svt1210t
		svt1210t.kdf
		svt1220
		svt1220.kdf
		svt1220a
		svt1220a.kdf
		svt1220a132
		svt1220c
		svt1220c.kdf
		svt1220c132
		svt1220t
		svt1220t.kdf
		svt1220t132
		termdef

End of Appendix A

Appendix B
MS-DOS ALLY Directory

This appendix lists the file contents of the release 2.00 ALLY directory, named "ally2_00." Note that while all ALLY files are in a single directory, they are separated by type here.

Type of File	File Name
* Development systems only	
AFILE	allyedit.a commen.h dialog.a*, dialog.e*, and dialog.h* errors.e hours.a start.a salary.fx employee.fx hours.fx
Executable	allydevx.exe* allyrunx.exe amigratx.exe* compactx.exe* dmigratx.exe* mergex.exe* mmigratx.exe newmsgx.exe* newprntx.exe newtermx.exe scriptx.exe*
Command file	acompact.bat* ally.bat allyedit.bat amerge.bat* amigrate.bat* ascript.bat dialog.bat* dmigrate.bat* install.bat

Type of File	File Name
* Development systems only	
	install2.bat mmigrate.bat newmsg.bat* newprint.bat newterm.bat
Format File	allyfmt
Object code for linking to 3GL driver program*	usercode.obj*
Printer definition file	DFLT printdef
C-code file for inclusion in driver program*	usercode.in*
Terminal definition file	pcterm pcterm.kdf termdef
Installation information file	devmsg1.doc devmsg2.doc devmsg3.doc readme.doc
System file	aconfig.sys sg_ask.com

End of Appendix B

Appendix C
ALLY Command Mnemonics

Mnemonic	Purpose
abortaction	Abort action
abortappl	Abort application
aborttask	Abort task
addnl	Add new line
bdelete	Back delete
bol	Beginning of line
bottom	Bottom
box	Box
budmode	Browse, update, delete mode
clrcasesens	Clear case sensitive
clrdrawmode	Clear draw mode
clrovertype	Clear overtype
clrpwrtype	Clear powertype
commit	Commit
compresswdw	Compress window
cpfrombuf	Copy from buffer
cptobuf	Copy to buffer
ctrlchar	Enter control character
definewdw	Define window
defmacro	Define macro
delbol	Delete to beginning of line
deleol	Delete to end of line
delline	Delete line
delrec	Delete current record
deltomark	Delete to mark
delword	Delete word
down	Down
downpage	Down page
dupdate	Deferred update

Mnemonic	Purpose
eol	End of line
exeman	Execute macro
exemacf	Execute macro from file
exitaction	Exit action
exitappl	Exit application
exittask	Exit task
expandwdw	Expand window
explodewdw	Explode window
fdelete	Forward delete
fhome	First field
find	Find
findanddel	Find and delete
finsnext	Insert first record in next group
flast	Last field
flistval	Move to list of values
fnext	Next field
fpickval	Pick from list of values
fprev	Previous field
frfunction	Invoke local function
glblreplace	Global replace
hightomark	Highlight to mark
hightypeset	Set highlight type
home	Home
homemch	Home area
ignore	Ignore
insafter	Insert record after
insbefore	Insert record before
insertline	Insert line
jumptomark	Jump to mark
khelp	Help
kmpprint	Print menu

Mnemonic	Purpose
ldtomark	Line draw to mark
left	Left
loadmacros	Load macros
macfmfile	Macro from file
mactofile	Macro to file
mark	Set mark
menu	Function key choice
movewdw	Move window
nextline	Next line
nextmch	Next area
nextword	Next word
overlayblk	Overlay block
pall	Print all
phome	First page
pickfield	Copy to field-buffer
picktask	Pick task
plast	Last page
pnext	Next page
ppage	Print page
pprev	Previous page
prest	Print rest
prevmch	Previous area
prevmenu	Previous menu
prevword	Previous word
prhome	First display area
prlast	Last display area
prnext	Next display area
prntscrn	Print screen
prntvnum	Print version number
prompt	Prompt line
prprev	Previous display area
putfield	Copy from field-buffer

Mnemonic	Purpose
qbe	Query by example
query	Execute query
qwhere	Query by where clause
readfile	Read from file
redraw	Redraw
refresh	Refresh
removeblk	Remove block
replace	Replace
resizewdw	Resize window
rghome	First group
rglast	Last group
rgnext	Next group
rgprev	Previous group
rhome	First record
right	Right
rlast	Last record
rnext	Next record
roamfirst	First area
roamlast	Last area
rollback	Rollback
rprev	Previous record
save	Save
savemacros	Save macros
scrollwdw	Scroll window
select	Choose from roam area
setcasesens	Set case sensitive
setdelaycnt	Pause
setdrawmode	Set draw mode
setovertype	Set overtype
setpwrtype	Set powertype
setrptcnt	Set repeat count
shell	Go to OS command line processor

Mnemonic	Purpose
task	Start task
terminator	Choose from prompt line
togcasesens	Toggle case sensitive
togdrawmode	Toggle draw mode
toggletask	Toggle task
togovertype	Toggle overtype
togpwrtype	Toggle powertype
top	Top
topmenu	First menu
turtleclear	Clear turtle
turtlehl	Highlight with turtle
turtleld	Line draw with turtle
uldtomark	Erase line draw
uldturtle	Erase line draw with turtle
unbox	Unbox
undelline	Undelete line
undelrec	Undelete record
undelword	Undelete word
up	Up
uppage	Up page
windone	Window-action
windown	Window down
winleft	Window left
winright	Window right
winup	Window up
writefile	Write to file

End of Appendix C

Appendix D
ASCII Character Codes

CTRL	Character	Binary Bit 7 to Bit 0	Octal	Decimal	Hexidecimal
@	NUL	00000000	000	000	00
A	SOH	00000001	001	001	01
B	STX	00000010	002	002	02
C	ETX	00000011	003	003	03
D	EOT	00000100	004	004	04
E	ENQ	00000101	005	005	05
F	ACK	00000110	006	006	06
G	BEL	00000111	007	007	07
H	BS	00001000	010	008	08
I	HT	00001001	011	009	09
J	LF	00001010	012	010	0A
K	VT	00001011	013	011	0B
L	FF	00001100	014	012	0C
M	CR	00001101	015	013	0D
N	SO	00001110	016	014	0E
O	SI	00001111	017	015	0F
P	DLE	00010000	020	016	10
Q	DC1	00010001	021	017	11
R	DC2	00010010	022	018	12
S	DC3	00010011	023	019	13
T	DC4	00010100	024	020	14
U	NAK	00010101	025	021	15
V	SYN	00010110	026	022	16
W	ETB	00010111	027	023	17
X	CAN	00011000	030	024	18
Y	EM	00011001	031	025	19
Z	SUB	00011010	032	026	1A
[	ESC	00011011	033	027	1B
\	FS	00011100	034	028	1C
]	GS	00011101	035	029	1D
^	RS	00011110	036	030	1E
_	US	00011111	037	031	1F
	SP	00100000	040	032	20
	!	00100001	041	033	21
	"	00100010	042	034	22
	#	00100011	043	035	23
	$	00100100	044	036	24
	%	00100101	045	037	25
	&	00100110	046	038	26
	'	00100111	047	039	27
	(	00101000	050	040	28
	)	00101001	051	041	29
	*	00101010	052	042	2A
	+	00101011	053	043	2B
	,	00101100	054	044	2C
	-	00101101	055	045	2D
	.	00101110	056	046	2E
	/	00101111	057	047	2F
	0	00110000	060	048	30
	1	00110001	061	049	31
	2	00110010	062	050	32
	3	00110011	063	051	33
	4	00110100	064	052	34
	5	00110101	065	053	35
	6	00110110	066	054	36
	7	00110111	067	055	37
	8	00111000	070	056	38
	9	00111001	071	057	39
	:	00111010	072	058	3A
	;	00111011	073	059	3B
	<	00111100	074	060	3C
	=	00111101	075	061	3D
	>	00111110	076	062	3E
	?	00111111	077	063	3F

Character	Binary Bit 7 to Bit 0	Octal	Decimal	Hexidecimal
@	01000000	100	064	40
A	01000001	101	065	41
B	01000010	102	066	42
C	01000011	103	067	43
D	01000100	104	068	44
E	01000101	105	069	45
F	01000110	106	070	46
G	01000111	107	071	47
H	01001000	110	072	48
I	01001001	111	073	49
J	01001010	112	074	4A
K	01001011	113	075	4B
L	01001100	114	076	4C
M	01001101	115	077	4D
N	01001110	116	078	4E
O	01001111	117	079	4F
P	01010000	120	080	50
Q	01010001	121	081	51
R	01010010	122	082	52
S	01010011	123	083	53
T	01010100	124	084	54
U	01010101	125	085	55
V	01010110	126	086	56
W	01010111	127	087	57
X	01011000	130	088	58
Y	01011001	131	089	59
Z	01011010	132	090	5A
[	01011011	133	091	5B
\	01011100	134	092	5C
]	01011101	135	093	5D
^	01011110	136	094	5E
_	01011111	137	095	5F
`	01100000	140	096	60
a	01100001	141	097	61
b	01100010	142	098	62
c	01100011	143	099	63
d	01100100	144	100	64
e	01100101	145	101	65
f	01100110	146	102	66
g	01100111	147	103	67
h	01101000	150	104	68
i	01101001	151	105	69
j	01101010	152	106	6A
k	01101011	153	107	6B
l	01101100	154	108	6C
m	01101101	155	109	6D
n	01101110	156	110	6E
o	01101111	157	111	6F
p	01110000	160	112	70
q	01110001	161	113	71
r	01110010	162	114	72
s	01110011	163	115	73
t	01110100	164	116	74
u	01110101	165	117	75
v	01110110	166	118	76
w	01110111	167	119	77
x	01111000	170	120	78
y	01111001	171	121	79
z	01111010	172	122	7A
{	01111011	173	123	7B
|	01111100	174	124	7C
}	01111101	175	125	7D
~	01111110	176	126	7E
DEL	01111111	177	127	7F

End of Appendix D

Index

End of Index