dL4

Installation & Configuration Guide

Unix

Revision 4.3

Information in this document is subject to change without notice and does not represent a commitment on the part of Dynamic Concepts Inc. (DCI). Every attempt was made to present this document in a complete and accurate form. DCI shall not be responsible for any damages (including, but not limited to consequential) caused by the use of or reliance upon the product(s) described herein.

The software described in this document is furnished under a license agreement or nondisclosure agreement. The purchaser may use and/or copy the software only in accordance with the terms of the agreement. No part of this manual may be reproduced in any way, shape or form, for any purpose, without the express written consent of DCI.

© Copyright 2001 Dynamic Concepts Inc. (DCI). All rights reserved.

Dynamic Concepts Inc.

18-B Journey

Aliso Viejo, CA 92656

www.dynamic.com

dL4 is a trademark of Dynamic Concepts Inc.

UniBasic is a trademark of Dynamic Concepts Inc.

Dynamic Windows is a trademark of Dynamic Concepts Inc.

BITS is a trademark of Dynamic Concepts Inc.

IRIS is a trademark of Point 4 Data Corporation.

c-tree is a trademark of Faircom.

IQ is a trademark of IQ Software Corporation.

UNIX is a registered trademark of UNIX Systems Laboratories.

SYBASE is a registered trademark of Sybase Inc.

CodeBase is a trademark of Sequiter Software Inc.

Microsoft, MS, MS-DOS, Microsoft Access, and FoxPro are registered trademarks, and ODBC, Windows and Windows NT are trademarks of Microsoft Corporation in the USA and other countries.

HOW TO USE THIS GUIDE 1

CHAPTER INTRODUCTION 1
GENERAL CONVENTIONS 1
KEYBOARD CONVENTIONS 1

INSTALLING DL4 RUNTIME 2

CHAPTER INTRODUCTION 2
LICENSING 2
COPYING dL4 INSTALLATION MEDIA 2
INSTALLING dL4 FROM DISKETTES OR TAPE 2
INSTALLING dL4 FROM AN INSTALLATION FILE 3

WORKING WITH DL4 CONFIGURATION OPTIONS 5

CHAPTER INTRODUCTION 5
CONFIGURING RUNTIME PARAMETERS 5
TABLE OF RUNTIME PARAMETERS 5
CONFIGURING A PRINTER 7
Configuring An Executable Pipe Script *8*
Configuring A Driver Indirection Script *10*
Configuring A Direct Output Script *10*
CONFIGURING THE TERMINAL 10
TESTING THE INSTALLATION 10

CONFIGURING THE OPERATING SYSTEM 12

CHAPTER INTRODUCTION 12
NUMBER OF PROCESSES 12
NUMBER OF OPEN FILES 12
MESSAGE QUEUES 12
SHARED MEMORY AND SEMAPHORES 15

INSTALLING DL4 DEVELOPMENT 16

CHAPTER INTRODUCTION 16
LICENSING 16
COPYING dL4 INSTALLATION MEDIA 16
INSTALLING dL4 FROM DISKETTES OR TAPE 16
INSTALLING dL4 FROM AN INSTALLATION FILE 17

WORKING WITH THE DL4 DEVELOPMENT KIT 18

CHAPTER INTRODUCTION 18
COMPILING/LINKING A CUSTOMIZED dL4 18
INSTALLING A CUSTOMIZED VERSION OF dL4 18

APPENDIX A: TERMINAL DESCRIPTION FILES 19

[OUTPUTMACROS] 19
[MACROS] 19
[OUTPUTUNICODEMAPPING] 19
[FUNCTIONKEYS] 20
[INPUTUNICODEMAPPING] 20
[INPUTACTIONS] 20
[SETTINGS] 20
TRANSLATION LANGUAGE STATEMENTS 20
EXAMPLES OF THE CONVERSIONS 21
[OUTPUTMACROS] SECTION 23
[OUTPUTMACROS] 23
[MACROS] SECTION 24

[OUTPUTUNICODEMAPPING] SECTION 25
[FUNCTIONKEYS] SECTION 25
[INPUTUNICODEMAPPING] SECTION 26
[INPUTACTIONS] SECTION 26
[SETTINGS] SECTION 27
SECTION ORDERING 28

How to Use This Guide

Chapter Introduction

This guide describes dL4 installation and configuration for UNIX users. It is assumed that the user is familiar with UNIX operating systems. The terms and conventions used in this guide are described below.

General Conventions

pfilter {**-c** *character-set*}	Values that you must supply are shown in italic type for clarity and to distinguish them from other elements of the syntax. In this example, *character-set* must be replaced by an actual character set name.
pfilter {**-c** *character-set*}	The right and left brace characters ({optional items}) indicate an item that is optional
WINDOW (**ON** | **OFF**)	Selection of one of a group of items is shown within parenthesis separated by |. Choose only one; in this example, the legal syntax is **WINDOW ON** or **WINDOW OFF**. The parenthesis is not part of the syntactical form.
`warning:`	Mono-spaced type is used to display screen output, keyboard input commands, filenames, and program examples.
OPEN #	Literal element of a command, a utility, or a statement and environment variables are shown in **bold** type.

Keyboard Conventions

The following conventions are used to define the keys and key combinations:

- Key names appear in capital letters and are referred to by their names only, without the word "key". For example, press **ESC** means press the key labeled "Esc".
- A plus sign (+) between key names means you hold down the keys in the order that they are listed and press the last key. For example, **CTRL+D** means hold down the **CTRL** and the **D** keys together.
- A comma (,) between key names means you press and release each key in turn. For example, "press **ESC**,**P**" means press **ESC** and release it then press **P** and release it.

Installing dL4 Runtime

Chapter Introduction

The dL4 runtime kit consists of scope, run, loadsave, and various supporting utilities and libraries. This chapter provides all of the information you need to install dL4 runtime on your hard disk from the installation media or from a downloaded installation file

Licensing

dL4 can only be installed and used with a valid license. Please refer to your dL4 license for terms and conditions in using dL4.

The Passport software is distributed by DCI as a separate installable product. The latest software versions may be obtained from our web site (www.dynamic.com) or from our ftp site (ftp.dynamic.com).

dL4 requires at least version 3.1 of the Passport daemon program in order to run. If you attempt to run dL4 with a daemon older than 3.1, you will see messages such as:

```
Waiting for response from security daemon ...
```

and eventually an exit.

Like DCI's other Unix products, dL4 requires a functioning Passport daemon to license the system and an SSN specific to that license number to authorize use of the software. In the absence of SSN authorization, a single-user demo mode is invoked.

Copying dL4 Installation Media

Before you install dL4 on your hard disk, DCI recommends making a copy of the installation media as a backup.

Installing dL4 from Diskettes or Tape

To install dL4 for UNIX, perform the following steps:

1. Log in as root.

Warning: Be careful. Do not issue any commands or perform any tasks other than the following instructions.

2. If upgrading an existing dL4 installation, use the UNIX **`ps`** or other utility to make certain that all dL4 scope and run users have exited dL4. If a program cache is used (see DL4CACHE in the chapter Working with dL4 Configuration Options), any existing cache must be deleted. Use the UNIX **`ipcs`** utility to find any shared memory or semaphores used by the cache name and delete them using the UNIX **`ipcrm`** utility.

3. You must be in the Bourne shell to install dL4. You can usually start a Bourne shell by typing:

```
/bin/sh.
```

4. Change to any temporary directory on your system by typing:

```
# cd /tmp
```

5. Place the dL4 media in the appropriate drive.
6. Use the command shown on the media label to load dL4 files to your disk. For example, if the format type on the label is cpio then you would type:

```
# cpio -imcduv </dev/{device_name}
```

   where *{device_name}* is the name of the tape or diskette drive on the UNIX system.

   If the cpio command is successful, a list of filenames is displayed as they are loaded into the installation directory.

   If the command shown is other than cpio, such as 'installpkg', follow the standard installation instructions of your UNIX system for installing software products.

   Print and read the file `readme.txt` from the installation directory. This is an ASCII text file and contains information pertinent to the current release of dL4.

   Store your original media in a cool, fireproof location.

7. Run the install script by typing:

```
# ./install
```

   This script moves the files to their proper locations in `/usr/bin` and `/usr/lib/dl4`. Install creates the directory `/usr/lib/dl4` if not already created, sets the file permissions, and deletes the temporary installation directory.

8. Exit from the root account and login as a normal user.

Installing dL4 from an Installation File

To install dL4 for UNIX, perform the following steps:

1. Log in as root.

Warning: Be careful. Do not issue any commands or perform any tasks other than the following instructions.

2. If upgrading an existing dL4 installation, use the UNIX **`ps`** or other utility to make certain that all dL4 scope and run users have exited dL4. If a program cache is used (see DL4CACHE in the chapter Working with dL4 Configuration Options), any existing cache must be deleted. Use the UNIX **`ipcs`** utility to find any shared memory or semaphores used by the cache name and delete them using the UNIX **`ipcrm`** utility.
3. You must be in the Bourne shell to install dL4. You can usually start a Bourne shell by typing:

```
/bin/sh.
```

4. Change to any temporary directory on your system by typing:

```
# cd /tmp
```

5. Copy the installation file to the installation directory by typing:

```
# cp /source_path/99_dl4_4.3.Z 4.3.Z
```

6. Uncompress the installation file by typing:

   ```
   # uncompress 4.3.Z
   ```

7. You now have an uncompressed file without the .Z in the name. Use cpio -imcduv with this file as the redirected input by typing:

   ```
   # cpio -imcduv <4.3
   ```

   Print and read the file `readme.txt` from the installation directory. This is an ASCII text file and contains information pertinent to the current release of dL4.

8. Run the install script by typing:

   ```
   # ./install
   ```

   This script moves the files to their proper locations in `/usr/bin and /usr/lib/dl4`. Install creates the directory `/usr/lib/dl4` if not already created, sets the file permissions, and deletes the installation directory.

9. Exit from the root account and login as a normal user.

Working with dL4 Configuration Options

Chapter Introduction

When dL4 is first installed, it uses a default configuration. This default configuration can be modified by defining and exporting environment variables. This may be done in the user's `.profile` file when the default value is insufficient.

Configuring Runtime Parameters

In order to execute, **scope** and **run** must locate a "terminal description file". The name of this file is taken directly from the **TERM** environment variable, and the following directories are searched, in order:

1. Directory given in **TERMDIR** environment variable, if present
2. Current working directory
3. Directory given in **HOME** environment variable, if present

The use of **TERMDIR** is the most practical for long-term setup. The stock terminal description files shipped with dL4 can be accessed by setting:

```
TERMDIR=/usr/lib/dl4/term
export TERMDIR
```

Table of Runtime Parameters

This section lists the user-configurable dL4 environment variables. When the default value is insufficient a new value may be assigned and exported to the environment from a user's `.profile` file so that the system can be tailored to the individual user's needs.

DL4CACHE	Name, default size, and default access rights of the program cache. The program cache allows different processes and users to share the same copy of a dL4 program or library and thus reduce memory usage. The **DL4CACHE** parameter is a string of the form “<access-permissions> [size] name”. The value of access-permissions is a standard dL4 file access option such as “<644>” or “<W>”. The size value is similar to the size value of a contiguous file and consists of a number of blocks followed by a colon and the block size in bytes. The name is a string or a Unix resource id in decimal (“nnn”), octal (“0nnn”), or hexadecimal (“0xnnnn”). If the size is specified, a cache will be created if it does not exist, otherwise the cache must be created by another user. If the cache does not exist, cannot be accessed, or cannot be created, the cache will be ignored. For highest security, most users should be given only read only access to a pre-existing and pre-initialized cache. A typical **DL4CACHE** value for a dynamic cache would be `"<666> [2048:4096] 0xdddc0500"`. For high security, the access option would be `“<644>”` for the user or startup program that initializes the cache and `“<444>”` for all other users. See the

	description of the ProgramCache intrinsic CALL in the dL4 Language Reference manual for additional information.
DL4PORTDUMP	Filename and path of the dump file used by the ForcePortDump and ProgramDump intrinsic CALLs. See the description of the ForcePortDump intrinsic CALL in the dL4 Language Reference manual for additional information.
ISAMFILES	Maximum number of opened Indexed file directories. For each indexed file opened, one entry is required for each Directory (index) plus 1 for the data file. The default value of 40 supports 10 indexed files open with an average of (3) directories (indices) each. If this value is too small, the error "`Internal error in driver`" or "`Driver resource exhausted`" is returned.
ISAMSECT	The size of Index file directory nodes in newly created Indexed files expressed in blocks of 128 bytes. The default value of 8 creates 1024 byte nodes which provide good performance under normal conditions. Larger nodes may be useful for extremely large (> 2 gigabyte) files or large keys.
ISAMMAXSECT	Maximum size of Index file directory nodes expressed in blocks of 128 bytes. If a file uses nodes larger than this value, the error "`Internal error in driver`" is returned when the file is opened. The default value is 16 (2048 bytes). This value should be greater than or equal to the value of the ISAMSECT parameter.
ISAMOFFSET	This parameter is used only with "uniBasic Indexed-Contiguous" files. It determines the displacement within records of the Deleted-Record-Flag and Delete-Link-List-Pointer. The default value of zero must be changed if your applications use "uniBasic Indexed-Contiguous" files and write data within the first 5 bytes of a record after it has been deleted.
LIBSTRING	Defines a set of paths to search for program filenames when linking, **CALL**ing, or **CHAIN**ing to a program. This value is displayed by the STATUS UNIT command in the debugger and returned by the built-in Basic **`MSC$(6)`** function. If this parameter is not defined, only the current working directory and the directory of the parent program are searched. The **LIBSTRING** value can be set at runtime by the **LIB** statement, but that value will only be used by the program that executed the **LIB** statement. **LIBSTRING** should be constructed to minimize number of searches required to locate programs.

LUMAP

Defines a list of directory mapping pairs which are applied to all relative filenames for both data and program files. The list is space seperated with each mapping pair consisting of a logical directory name, an equals sign ("="), and an absolute path. When a relative filename is used and that filename begins with a logical directory defined in LUMAP, then the logical directory will be replaced by the absolute path from LUMAP.

For example, given an LUMAP value of "5=/usr/accounting Mail=/disk2/Mail", the following filenames would be mapped as shown:

```
5/filename    -> /usr/accounting/filename

Mail/filename -> /disk2/Mail/filename

x/5/filename  -> x/5/filename (unchanged because"x"
isn't in LUMAP)
```

MAXPORT

Defines the maximum port number. The default value of **MAXPORT** is 4095. The value of **MAXPORT** is used for automatic port number assignment by the **SPAWN** statement and during sign on to assign the **PORT** number of a dL4 session when **PORT** is undefined. Since dL4 assigns port numbers in a

	decreasing order from **MAXPORT**, it may be used to set unique port numbers on different systems connected via a network.
MSC7	Defines the numeric value to be returned by the MSC(7**)** function. If **MSC7** is not defined as an environment variable the value of 257 is returned.
PORT	Forces the current session to operate as a specific **PORT** number, e.g. **PORT**=23. The maximum **PORT** number must be a value between 0 and **MAXPORT**. If **PORT** is not defined, a port number will be assigned, if possible, by examining the user terminal device name (for example, "/dev/tty42" would default to port 42).
SPC5	Defines the numeric value to be returned whenever the **SPC**(5) function is called. If **SPC5** is not defined as an environment variable the value of 257 is returned.
SPC7	Defines the numeric value to be returned whenever the **SPC**(7) function is used. If **SPC7** is not defined as an environmental variable, zero (0) is returned.
TERM	Defines the terminal type. **TERM** will be used as the name of the terminal definition file loaded by the terminal driver.
TERMDIR	Defines the path of the directory containing the terminal definition files used by the terminal driver. If **TERMDIR** is not defined or if the file selected by the **TERM** environment variable does not exist in **TERMDIR**, then the terminal definition file will be loaded from the current working directory or the directory specified by the **HOME** environment variable.
WINDOWDEFAULTS	The "**FG**" and "**BG**" RGB values defined by this parameter control the default foreground and background colors in dL4 windows. For example, if **WINDOWDEFAULTS** is set to "fg=16776960,bg=0", then a newly opened dL4 window will use yellow (RGB value = 16776960 = 255 * 65536 + 255 * 256) as the foreground color and black (RGB value = 0) as the background color. If the **WINDOWDEFAULTS** environment variable is not defined or if it doesn't contain foreground or background colors, then the default foreground color is black and the default background color is white. If a terminal definition file supports color, but does not support the **'BACKCOLOR'** mnemonic, the **WINDOWDEFAULTS** environment variable should be set to use the actual background color used by the terminal.

Configuring A Printer

A dL4 program accesses printers by opening a filename with a leading dollar sign such “$printer1”. Such opens are directed to the pipe driver which selects the actual driver used to perform the open. A printer script is a file, an executable shell script in most cases . The script filename should not begin with a dollar sign ("$"); the dollar sign used in the **OPEN** statement is a special character that tells the dL4 open routine that this is a script open. Unless the printer is opened with an absolute path (“$/usr/lib/dl4/lpt”), a printer script file must be in a directory specified in the **PATH** variable.

There are three types of printer scripts:

1. executable pipe scripts - these scripts contain executable commands and are executed as separate processes communicating with the dL4 program via pipes.
2. driver indirection scripts - these scripts redirect output to a specific driver and effectively convert any open of the script to an **OPEN AS** statement using filename and option parameters included in the script.

3. direct output scripts - these scripts are used to output to physical devices such as serial ports using the translation features of a Terminal Definition File.

Most Unix printer scripts are executable pipe scripts. Note that of all three types only an executable pipe script is actually executed as shell script.

Printer script options are specified by the first line of the script if it begins with the string “# dl4opts=” (case insensitive). If the first line does not begin with “# dl4opts=”, the script will be treated as an executable pipe script using the UniBasic-ASCII character set. The options determine the type of script and its features. Each option consists of a case insensitive option name followed by an equals sign and then the option value. Spaces are allowed in the option value. All options are comma separated.

Example:

```
# dl4opts=charset=UTF-8,lock=true
```

The following options are supported by the pipe driver:

Option Name	Value	Meaning
buildas	driver name	Driver to be opened in build mode by a driver indirection script
charset	character set name	Character set of pipe to an executable pipe script
format	“true” or “false”	If “false”, TAB and PRINT COMMA mnemonics will be passed unchanged to the executable pipe script (default is “true”). This option should be used if the printer script implements the ‘BC’ (Begin Compressed) and ‘BX’ (Begin eXpanded) mnemonics. If format is false, the executable script must implement the “@x’ and ‘ALIGN’ mnemonics.
lock	“true” or “false”	If “true”, the script can only be used by one user and on one channel at a time. Once open, subsequent opens will return a “device open elsewhere” error until it is closed.
openas	driver name	Driver to be opened by a driver indirection script
options	driver option	A driver option to be passed down to the driver used by a driver indirection script
output	device or file name	Device or file name to be opened by a direct output script
path	device or file name	Path argument to be passed down to the driver used by a driver indirection script
translate	file path	Path of printer or terminal definition file used by a direct output script.

In driver indirection scripts or direct output scripts, all unrecognized options are passed as options to the driver selected by the script. The “options=” option can be used to pass recognized options such as “charset=” to the lower level driver.

Configuring An Executable Pipe Script

A dL4 program normally uses printers by opening a pipe to an external program, typically a bourne shell script file, and then outputting to that pipe. For example, the statement ‘**OPEN #2,**"$printer1"’ would start up a separate process to run the script file `printer1` and send all output from channel 2 to be filtered by `printer1` script. The script is responsible for reading dL4 program output from standard input, translating the input characters into the form required by the printer, and then sending the translated characters to the printer.

The following is a sample printer shell script file that outputs to a device:

```
# dL4opts=charset=utf-8,lock=/tmp/lpt2.lk
set TERMDIR=/usr/lib/dl4/printers
pfilter -c utf-8 pclprinter >/dev/lp00
```

The first line of this script file is an options line interpreted by the dL4 pipe driver. This options line specifies that the pipe driver should send all output to the script file in the UTF-8 character set. It also specifies that the printer should be locked so that any subsequent attempt to open the printer will cause an "`device is open elsewhere`" error until the printer is closed. To use pipe driver options the first line of the script file must begin with "`# dL4opts=`". Options use the format "*`keyword=value`*". The supported keywords are "`charset`" and "`lock`":

`charset` is used to specify the character set used by pipe I/O. The available character sets and their synonyms are:

"US-ASCII" or "ASCII" or "ISO 646"
"ISO 8859-1" or "ANSI" or "ANSI Latin 1"
"IRIS" or "IRIS-ASCII"
"UniBasic" or "UniBasic-ASCII"
"IBM Code Page 437"
"IBM Code Page 850"
"Windows" or "Windows Code Page 1252"
"EBCDIC" or "EBCDIC 037"
"UTF-8"

The most flexible character set is UTF-8 which is a multibyte encoding of Unicode, the native character set of dL4. Using UTF-8 allows the pipe driver to pass any character used by a dL4 program to the printer script. UTF-8, however, can only be used with script files that can accept UTF-8. The default character set is "UniBasic-ASCII" which consists of the ASCII character set and the UniBasic mnemonics.

lock can be used to prevent concurrent opens of a printer script. The value of "lock" specifies the path of a lock file to be built by the script, so that only one simultaneous open to the printer will be allowed.

It's also possible to output to a spooled printer. The following printer shell script file outputs to the spooled system printer "lp1":

```
# dL4opts=charset=utf-8
set TERMDIR=/usr/lib/dl4/printers
pfilter -c utf-8 pclprinter | lp –d lp1
```

Note that this script does not use the "lock" option because spooled printers do not need locking.

The sample script files use the dL4 pfilter utility to translate the piped output into the form required by the printer. The pfilter utility is a program with the following command line syntax:

```
pfilter {-c character-set} {-d devicename} printer-type
```

Where:

`character-set` specifies the input character set (default is UniBasic ASCII). The supported character sets are the same as those listed above for the "`charset=`" script file option.

`devicename` can be any printer device available to the system. If the "-d *devicename*" option is not specified, pfilter will output to standard output.

`printer-type` is the path of a printer definition file, which is similar to a Terminal Definition File (TDF). A printer definition file controls the translation of the character's output by dL4 to the characters required by the printer. The printer definition file should be specified by an absolute path (such as "`/usr/lib/dl4printers/hplj`") or be in the directory specified by the **`TERMDIR`** environment variable. The **`TERMDIR`** environment variable can be set in the printer script file as in the example above.

The format of printer definition files is identical to dL4 for UNIX terminal definition files except that only the output sections are required. See the documentation on the Terminal Description Files for more information. A simple printer definition that copies all ANSI Latin 1 characters without translation would appear as follows:

```
[OutputMacros]
Default=%@%c
[OutputUnicodeMapping]
; This mapping is set up for 8-bit output on a printer using
; the ISO 8859-1 (i.e. ANSI Latin 1) character set.
```

```
Set0=
0x0000-0x00ff=0x00
```

Configuring A Driver Indirection Script

A driver indirection script causes the pipe driver to open a script specified driver and to pass all I/O operations to that driver. A driver indirection script consists of a single line beginning with "# dl4opts=" (case insensitive) and containing a "buildas=" or "openas=" option to select the I/O driver. All subsequent lines are ignored and treated as comments. A "path=" option can be used to pass a filename argument to the selected driver. All options not recognized by the pipe driver are passed to the driver selected by the "openas=" or "buildas=" value.

The driver indirection script below opens serial port /dev/tty3 as a printer at 19200 bps, 8 data bits, no parity, 1 stop bit using default output translation. A "term=" option could be added to select a terminal definition file.

```
# dl4opts=openas=Serial Terminal,path=/dev/tty3,speed=19200,data=8n1
```

Configuring A Direct Output Script

A direct output script causes the pipe driver to open a driver stack consisting of the terminal translation driver and the raw file driver. All output is passed to the terminal translation driver which uses a terminal definition file to translate characters and mnemonics from Unicode to the character sequences needed by the output device. The translated characters are then output through the raw file driver. A direct output script consists of a single line beginning with "dl4opts=" (case insensitive) and containing both "output=" and a "translate=" options. All subsequent lines are ignored and treated as comments. The "output=" option selects the device to be opened by the raw file driver while the "translate=" option specifies the path of the terminal definition file. All options not recognized by the pipe driver are passed to the raw file driver.

The direct output script below opens serial port /dev/tty3 as a printer at 19200 bps, 8 data bits, no parity, 1 stop bit. The printer definition file "pcl" is used to translate character and mnemonics from Unicode to the character sequences needed by the printer.

```
# dl4opts=output=/dev/tty3,translate=/usr/dl4/pcl,speed=19200,data=8n1
```

Configuring the Terminal

The stock terminal description files shipped with dL4 are in directory /usr/lib/dl4/term. If you have a terminal other than any of the types supplied by the dL4 package you must create your own terminal description file in the above directory. Refer to Appendix A for information on the terminal description file. It is recommended that you copy an existing terminal description file to a file with the name of your terminal and modify this file as needed. Remember, this name must be in the **TERM** environment variable.

Testing the Installation

Login as a normal user and verify that /usr/bin is part of your path. Verify that the environment variable **TERM** is set to the correct terminal description file name and dL4 can find it by one of the methods described in section Configuring Runtime Parameters. At the system prompt type:

```
$ scope
```

The Scope Command Interpreter version, copyright notice, and user license information should be displayed. The prompt should change from a $ to a # and dL4 is ready to execute your commands. To exit type:

```
#bye
```

dL4 is now installed!

Configuring the Operating System

Chapter Introduction

Like all Unix programs, dL4 uses operating system resources when it runs and as it accesses files. These resources are allocated dynamically as needed in some Unix systems, but on others, fixed amounts of each resource are configured into the operating system. On systems with fixed allocations, if may be necessary to increase the current or default resource limits to support the licensed number of users. This chapter describes what sort of resources may need reconfiguration. For instructions on how to reconfigure the operating system limits, please see your operating system documentation.

Number of Processes

Many Unix systems limit the total number of processes that can exist simultaneously. Each running instance of dL4 scope or run is a process. Additional processes can be created by the **SPAWN**, **PORT**, or **CALL TRXCO()** statements. Each time a printer is opened, the printer script may create several processes until the printer is closed. For most systems, the recommended number of processes is 5 times the number of users. Before increasing the maximum number of processes allow, please check your operating system documentation for other associated resources that will also have to be increased. If many or all users log on the system with the same used id, it be necessary to increase the maximum number of processes allowed per user.

Number of Open Files

The operating system must be configured to support the total number of channels opened to files or physical devices by all users. Some file types, such as Indexed Contiguous or Full-ISAM files, consist of two files and so opening a single dL4 channel will actually open two files. Most data files also use one operating system record lock resource for each open file and another record lock resource for each locked record.

As an example, consider an 8 user system. Each user might have 10 file channels open and, assuming Indexed Contiguous files, up to 20 operating system channels open. The system would have to be configured to support at least 160 (8 * 10 * 2) open files. In addition, the system would have to support at least 80 (8 * 10) and up to 160 (8 * 10 * 2) record locks. This does not include the number of files opened and record locks used by operating system itself. On some systems, it will also be necessary to configure the maximum number of unique file opens (where the same file opened by different users counts as one file) or the amount of file buffer space (this is very important for system performance).

Message Queues

For all inter-process communication, dL4 relies on Unix message queues. A message queue is created at startup to transmit and receive data between users. Such messages include:

- SIGNAL 1 & 2 and SEND/RECV data between ports
- CALL TRXCO and PORT statement commands and status
- Security communications

On most systems, the Unix command **ipcs** may be used to display information about message queues. Each message queue is identified by a unique 32-bit number, usually displayed as an 8-character hexadecimal value.

DCI products are identified by our own numbering scheme, which when viewed in hexadecimal, takes on an appearance such as DC00pnnn. The digits correspond to:

DC	**Dynamic Concepts Product**
00	Always zero
p	DCI Product ID:
0	Passport daemon
1	UniBasic IRIS
4	IQ
5	dL4
nnn	dL4 port number, in hexadecimal, associated with this queue. For example, port 15 is displayed as "00F".

Message queue requirements for dL4 are based on the number of concurrent users and overall message traffic on the system. The default values on many systems are sufficient to support a few users, but certainly will need to be increased for large installations. If they are not configured, dL4 may fail at start-up and output an error message.

The following 7 parameters affect message queues on most systems. The actual parameter names may vary:

MSGMNI	Maximum number of message queues. Configure based upon the maximum number of concurrent dL4 users plus phantom ports plus other DCI products such as IQ for Unix users plus one for the passport security daemon.
MSGMAX	Maximum size of a message in bytes; at least 516.
MSGMNB	Maximum number of bytes per message queue. Set to the maximum allowable value; typically 32768.
MSGTQL	Maximum number of outstanding system wide messages. Suggested setting is at least 256, but may be adjusted if message activity is known to be greater or smaller.
MSGSSZ	Size (in bytes) of a message segment. Memory for message data is divided into segments of the defined size. A value of 32 is recommended.
MSGSEG	Number of message segments within the system. MSGSEG * MSGSSZ determines the total number of bytes reserved for message data. The recommended formula is MSGSEG = (MSGTQL * 512)/MSGSSZ. For 256 dL4 concurrent messages, the value would be: (256 * 512) / 32 = 4096.
MSGMAP	Number of entries in the message map table. Each entry represents a contiguous free area in the message segments. The recommended formula is MSGMAP = MSGSEG/8 which, using our example, would be 512. If dL4 reports "Communication buffer is full" when the actual number of outstanding messages is < MSGTQL, first increase MSGMAP. If that doesn't correct the error, increase MSGSEG.

AIX Note: There are no user-configurable message queue parameters on AIX. The parameters are hard-coded in the kernel, and seem adequate for most installations.

SCO OpenServer Note: the message queue parameters in SCO OpenServer cannot be configured using the normal interactive utilities. Instead, the **idtune** command line utility must be used.

The following points must considered during configuration:

- Free message space <u>must</u> be available on the system. If the queues become full, additional users, including phantom ports, cannot be launched into dL4 or IQ. In addition, existing users may be prevented from performing **SPAWN**, **CALL TRXCO,** and **PORT** statements.
- A processes queue and any waiting messages are deleted if and when the port exits normally. If a process is killed, it cannot delete its queued messages.
- The configuration guidelines shown above consider only dL4 requirements. They do not include requirements of other Unix applications which rely on message queues.

Shared Memory and Semaphores

If the dL4 program cache (see **DL4CACHE** above) is used, the operating system must be configured to support shared memory and semaphores. A single shared memory segment and a single semaphore are shared by all users of a program cache. The operating system may also have configurable limits on the maximum size of a shared memory segment and on the maximum number of users of a single semaphore (this is often named **SEMMNU**, the maximum number of “undo” operations).

Installing dL4 Development

Chapter Introduction

The dL4 development kit provides the files necessary to build a customized version of dL4 with user written intrinsic calls and drivers. This chapter provides all of the information you need to install dL4 development on your hard disk from the installation media or from a downloaded installation file.

Licensing

dL4 can only be installed and used with a valid license. Read your dL4 license for terms and conditions in using dL4.

Copying dL4 Installation Media

Before you install dL4 on your hard disk, DCI recommends making a copy of the installation media as a backup.

Installing dL4 from Diskettes or Tape

To install the dL4 for UNIX development kit, perform the following steps:

1. Log in to account that will be used for development.
2. Create a directory to contain the development kit by typing:

   ```
   # mkdir dl4dev
   ```
3. Change to the directory by typing:

   ```
   # cd dl4dev
   ```
4. Place the dL4 media in the appropriate drive.
5. Use the command shown on the media label to load dL4 files to your disk. For example, if the format type on the label is cpio then you would type:

   ```
   # cpio -imcduv </dev/{device_name}
   ```

   where *{device_name}* is the name of the tape or diskette drive on the UNIX system.

   If the cpio command is successful, a list of filenames is displayed as they are loaded into the directory.

   If the command shown is other than cpio, such as 'installpkg', follow the standard installation instructions of your UNIX system for installing software products.

   Store your original media in a cool, fireproof location.

Installing dL4 from an Installation File

To install the dL4 for UNIX development kit, perform the following steps:

1. Log in to the account that will be used for development.
2. Create a directory to contain the development kit by typing:

   ```
   # mkdir dl4dev
   ```

3. Change to the directory by typing:

   ```
   # cd dl4dev
   ```

4. Copy the installation file to the installation directory by typing:

   ```
   # cp /source_path/99_dl4dev_4.3.Z 4.3.Z
   ```

5. Uncompress the installation file by typing:

   ```
   # uncompress 4.3.Z
   ```

6. You now have an uncompressed file without the `.Z` in the name. Use the command "`cpio -imcduv`" with this file as the redirected input by typing:

   ```
   # cpio -imcduv <4.3
   ```

   The files are moved to their proper locations in `dl4dev`.

Working with the dL4 Development Kit

Chapter Introduction

The purpose of this chapter is to briefly introduce the developer to using the dL4 development kit. This chapter does not explain how to write or debug an intrinsic call or a driver.

Compiling/Linking a Customized dL4

The dL4 development kit is used to add user written intrinsic **CALL**s, intrinsic functions, or drivers to dL4.

Note: Both intrinsic **CALL**s and functions are written in C language.

The kit can also be used to recompile the standard drivers in order to change standard options (see the individual driver sources for more information on such options). The development kit consists of the object files, sample source files, and make control files necessary to build custom versions of dl4.

To add an intrinsic call or function, you must copy the source files for the new call to the "dl4/devel/usercall" directory and add the new call to the files: Makefile, userproc.c, and userproc.h (these files are in the "usercall" directory). Examine the entries for existing calls to determine what changes to make.

To add a driver, you must copy the source files for the new driver to the "dl4/devel/driver" directory and add the new driver to the files: Makefile, driver.c, and drvrlib.h (these files are in the "driver" directory). Examine the entries for existing drivers to determine what changes to make. To modify a driver, simply edit the source file.

After making the required modifications, the appropriate dl4 executables must be recompiled and relinked. The current procedure to compile and link the libraries is described in the development kit readme file. This readme file is the text file "dl4/devel/readme.txt".

Installing a Customized Version of dL4

To install a customized version of dL4, install a standard version of dL4 and then replace the appropriate dL4 executables in the "/usr/bin" directory with the customized versions produced in "dl4/devel/" directories by the development kit. Please note that the UNIX operating system will not allow copying of the files if any of them are currently in use. If dL4 or a dL4 utility is executing, an error will occur if you attempt to delete or replace the current executables.

Appendix A: Terminal Description Files

The purpose of a terminal description file ("TDF") is to allow dL4 programs and dL4 itself to be terminal and printer independent. Instead of outputting terminal dependent sequences of control characters, dL4 programs output Unicode characters and mnemonics that can be used on any terminal or printer. The TDF is then used to translate the device independent Unicode characters into the specific control characters and escape sequences needed by a particular terminal or printer. For example, a dL4 program can output the Unicode copyright character ("©" or "\x00a9") and the terminal driver, using the appropriate TDF, will output the characters needed to select the necessary character set in the terminal and display the copyright character. Similarly, a dL4 program can output a clear screen mnemonic, 'CS', and depend on the TDF to generate the escape sequence needed by whatever terminal is currently being used.
When the dl4 terminal driver is opened at dL4 startup, it must locate and open a TDF. The name of the file exactly matches the value of the TERM environment variable. Assuming that your TERM variable is "ansi", the driver will look for a text file named "ansi" in the following directories:

- The directory specified by the TERMDIR environment variable
- Your current working directory
- The directory specified by the HOME environment variable

Terminal definition files are also used by the pfilter utility to translate characters sent to printers. When used by pfilter, a terminal definition file will often be called a printer definition file. The file syntax and use of the TERMDIR environment variable is identical.
The terminal description file is a text file that can be edited with any text file editor. Any line beginning with a semicolon will be treated as a comment line. Each TDF is composed of sections, each controlling a different aspect of terminal I/O processing. A section is delimited by a line containing the section title enclosed in square brackets. There are as many as seven sections:

[OutputMacros]

This **required** section allows the definition of **output** characters to be translated by the terminal driver. Terminal initialization is also defined in this section. This section defines the mnemonics associated with dL4 capabilities. The mnemonics are used by applications to control the CRT, keyboard and auxport, cursor behavior, and color output.

[Macros]

This optional section allows the programmer to design generic macro subroutines which can be referenced from the **[OutputMacros]** section.

[OutputUnicodeMapping]

This **required** section allows definition of the **output** mapping of 16-bit Unicode characters to equivalent characters for the terminal device, typically 7-bit or 8-bit codes. This mapping is performed after any translation by the **[OutputMacros]** section. This table controls the final character output to the terminal screen. The programmer is able to define multiple character sets accommodated by the terminal, typically used to output non-English characters, and line graphics.

[FunctionKeys]

This optional section allows the definition of **input** character strings to be translated to single Unicode characters by the terminal driver. Translation strings are defined as literal Unicode strings expected as input.

[InputUnicodeMapping]

This optional section allows definition of the **input** mapping of received terminal characters to equivalent 16-bit Unicode characters. This table controls the initial character input from the terminal keyboard except for those character strings processed by the **[FunctionKeys]** section.

[InputActions]

This optional section defines which specific input **actions** are to be performed for each translated Unicode character accepted as input. The input **action** determines whether an input is treated as data, as an editing key, or as a special action key.

[Settings]

This optional section contains miscellaneous parameter settings.
The **[OutputMacros]** and **[OutputUnicodeMapping]** sections are required, the others are optional.
When writing a **TDF**, it may be helpful to think of output as being originally in Unicode, passed first through the **[OutputMacros]** section (which is a Unicode to Unicode translation), the result of that translation being passed through the **[OutputUnicodeMapping]** section (which is a Unicode to 8-bit byte stream translation), and then the 8-bit byte stream being output to the terminal or printer. Similarly, each 8-bit input character is passed first to the **[FunctionKeys]** section (a byte string to Unicode character translation) which either recognizes it as part of a function key sequence or passes it on to the **[InputUnicodeMapping]** section (an 8-bit byte to Unicode character translation). After being translated to Unicode by either the **[FunctionKeys]** or the **[InputUnicodeMapping]** section, the input character is processed by the **[InputActions]** section (a translation of individual Unicode characters that pairs each Unicode character with an input action).

Translation Language Statements

The **Translation Language** statements constitute a small programming language similar in design and purpose to terminfo in Unix. These statements are used in the translation strings of the **[OutputMacros]** or **[Macros]** sections of the **TDF**. Many of these commands operate on a numeric stack which can be used for general numeric calculations. The stack can only push or pop one value onto or off of the stack at a time. For example, %Sx will Set the variable "x" to what ever numeric value is at the top of the stack and then pop that value off the stack. %Gx will Get the value of the variable "x" and push it onto the stack. Each of the commands must have a % preceding the actual command. Various arithmetic, logical, or bit operations can be performed on these stack values. When an operation, such as addition, with two operands is performed, two values are popped from the stack and the result is then pushed onto the stack, replacing the two previously popped items. For example, if x=6, y=2, the macro **-** '%Gx%Gy%-' would push the value of "x", 6 onto the stack, then the value of "y", 2, onto the stack, and finally the '-' operation would pop each value off and push the result. Thus, 2 would be popped, then 6, and the two values would then be subtracted (6 - 2), returning the result 4, which would then be pushed onto the stack.

%%	output %
%[[:]flags][width[.precision]][doxX]	

Pop an integer value and then output it formatted as in the C print function, flags are [-+#] and space.
The - flag Left justified within the specified field. Padding will be placed to the right of the value within the field. This flag is relevant only when a field size is specified and the output value is smaller than the minimum width.
The + flag Will produce an explicit + sign before all positive values. (Negative values are always preceded by **-** regardless of whether a plus-sign is specified.) This flag is relevant only for the conversion operation ***d*** .
The # flag This flag is used with the ***o***, ***x***, and ***X*** conversion operation. With this flag an alternate form of the main conversion is used. (e.g. %:#06x if x = 45 this will produce 0x002d.)
Width Width is a decimal digit string specifying a minimum field width. If the converted value has fewer characters than the field width, it is padded on the left (or right, if the left adjustment flag "-" has been specified). The adding is done with spaces unless the first character of width is a zero, in which case the padding is done with zeros.
.**precision** Precision is a decimal digit string succeeding a dot "." specifying the minimum number of digits to appear for the "d", "o", or "x" conversions.
The <space> flag Precede the result of a signed numeric conversion with a space or minus sign. This flag is ignored if the plus sign flag + is given.
The d conversion This will output a decimal integer. (e.g. %d if value = 45 then the output will be the character string "45").
The o conversion This will output an unsigned octal integer. (e.g. %o if value = 45 then output = "55")
The x and X conversion This will output an unsigned hexadecimal integer displayed with a lowercase ("x") or an uppercase ("X"). (e.g. %x%X if value = 45 then output will be "2d2D").

Examples of the Conversions

Sample format	Sample output Value = 45	Sample output Value = -45
%:#012x	0x000000002d	0x00ffffffd3
%:-d	45	-45
%:+d	+45	-45
%:-+d	+45	-45
%:-12.4d	0045	-0045

%*c*	pop an integer value and then output that integer as a character.
%"c"	push the integer value of the character "*c*" onto the stack
%@	push the integer value of the original (untranslated) character onto the stack. The original character is the current character being translated in the **OutputMacros** definition.
%p[1-9]	push *n*th parameter. A parameter is an argument that is being passed to a mnemonic. For string parameters, this will push the index of the string parameter. Example: in '10CR', '10' is the first, and only, parameter. The operator %P1 will push the value 10 onto the stack.
%{*nn*}	push numeric constant *nn (decimal),* 0*nn (octal),* or 0x*nn (hexadecimal). F*or example, %{13}, will push 13 onto the stack.
%E{ *var* }	push numeric value of Environment variable *var*. 0 (zero) is pushed if the Environment Variable is undefined, or contains non-numeric data. May be decimal ("*nnn*"); octal ("0*nnn*"); or hex ("0x*nnn*"). An example of a *var* is 'COLS' and 'LINES'
%S*x*	set variable *x* to pop(); [A-Z] = driver variables, [a-z] = user variables. User variables need not be predefined. The top item is popped from the stack and the value is stored into a user variable, [a-z]. Driver variables are used by the driver and are predefined. Stack size becomes stack size - 1
%G*x*	get variable *x* and push(). This will read the value of any variable (driver or user defined), and push that value onto the stack. Stack size becomes Stack size + 1

%& %| %^	bit operations: i = pop(), push(pop() op i); And, Or, Exclusive Or. The first two values at the top of the stack are popped and the designated operation is performed between the bits of the values and the resulting value is then pushed onto the stack
%~	bitwise negation operation (unary): Not. push(op pop()). Every bit in the binary representation of ~x is the inverse of what it was in the converted operand x. (e.g. binary: x = 1111000011110000, then ~x has the value 0000111100001111).
%!	logical negation operation (unary): push(op pop()). Computes the logical negation of its operand. (e.g. In C, the expression !(x) is identical to (x)==0).
%= %> %<	comparison operations: i = pop(), push(pop() op i). The first two values at the top of the stack are popped and the designated operation is performed and the resulting boolean value is then pushed onto the stack. (e.g. %{7}%{5}%<%d This will pop 7, pop 5, compare the two (7 < 5 = 0), and the push the result of 0 (false). If the result is true then the output is 1)
%A %O	logical operations: and, or. The first two values at the top of the stack are popped and then result is then pushed onto the stack. Any non-zero value is treated as true (1). The result is either true (1) or false (0).
%i	add one to the first and second (if any) parameters; useful for ANSI terminals
%+ %- %*%/ %m	arithmetic (%m is mod): i = pop(), push(pop() op i). The first two values at the top of the stack are popped and the designated operation is performed and the resulting value is then pushed onto the stack. (e.g. x = 1020, y = 128; %Gx%Gy%/%c this will push x (1020), push y (128) and then pop y, pop x, and then perform the division operation (1020 / 128 = 7.96875). Since the value is truncated, then value "7" will be pushed onto the stack.
%M*x*	perform macro x; x = [a-z]
%r *expr* %;	repeat expression pop() times (e.g. %{4}%"a"%Sx%r%Gx%c%Gx%{1}%+%Sx%; This will start a loop with a counter of 4 on the stack and the character "a" in variable x. The loop will repeat four times with each iteration outputting the character in variable x and then incrementing "x". Thus the string "abcd" will be output.
%? *expr* %t *thenpart*	
%e *elsepart* %;	If-Then-Else: %e *elsepart* is optional; Else-If conditions are allowed (e.g. x = 20, y = 50; %**?**%Gx%Gy%<%t%Sx%e%Sy If x < then set x else set y)
%$”cc”	Push string literal “cc” onto the string stack and push the index of that string onto the numeric stack.
%$D	Delete the string on top of the string stack.
%$c	Pop an integer value *i* from the numeric stack and then push the *i*’th character of the string stack onto the numeric stack.
%$l	Pop an integer value *i* from the numeric stack and then push the length of the string at the *i*’th character of the string stack onto the numeric stack.
%$F	Compare strings ignoring case. The top of the numeric stack (TOS) must be the starting index of the target string in the string stack. The top minus one of the numeric stack (TOS-1) must be the number of strings to compare the target against. The strings to be compared against must be defined by pairs of numbers on the numeric stack. The first string must have its starting index at TOS-3 and a related integer at TOS-2. The other strings must be identified by similar pairs on the stack. If the target string matches one of the strings, the related integer will returned on the stack. If no match is found, a zero will be returned on the stack. All parameters of the operator will be popped from the numeric stack (the string stack is not changed).

Standard driver variables:

C	Number of columns
R	Number of rows
X	Current column
Y	Current row
Z	Terminal type number

[OutputMacros] Section

The **[OutputMacros]** section of the **TDF** allows the definition of characters to be translated by the terminal driver when output. The left side of the equation specifies a set of Unicode characters; the right side gives the translation string to be executed in order to render the character(s) (or perform the function) on the terminal device. If a character is a member of the specified set of Unicode characters, then any output of that character will be performed by first evaluating the translation string.

A set of Unicode characters is defined by a comma-separated list of character or character range specifications. A range is defined by two character specifications separated by a hyphen (-). A character specification may be one of the following:

- A mnemonic value from dL4 BASIC, enclosed in single-quotes, e.g. 'CS' or 'U+1234'. The mnemonic letters may be preceded by one or more pound signs (#), separated by commas, to indicate that this translation only applies when the specified number of character parameters are present. For example, the entry for '#CR' applies when a BASIC program outputs '10CR'.
- The special character @ followed by a list of one or more pound signs (#). @#,# is an abbreviation for to '#,#MOVETO'.

The translation strings themselves are defined as literal Unicode strings to output embedded with optional translation statements. Each string is a list of one or more of the following:

^*x*	Control-*x* for any appropriate *x*
\E or \e	ESCAPE
\r	CARRIAGE RETURN
\n or \l	LINE FEED
\t	HORIZONTAL TABULATION
\v	VERTICAL TABULATION
\f	FORM FEED
\s	SPACE
\a	BELL
\^	Literal ^
\'	Literal '
\"	Literal "
\\	Literal \
\ooooo	Unicode character value in octal
\x*hhhh*	Unicode character value in hexadecimal
%...	Translation statement.

Three special macros exist in this section; “Init”, "Close", and “Default”. Any output characters not explicitly defined here are output using the “Default” translation, which typically should be defined to just output the character as-is and increment the column counter. The “Init” string is executed once at driver initialization time, and typically would initialize the terminal and any variable values. The "Close" string is executed when the driver is closed and can be used to clear the screen when dL4 exits or output a formfeed when the pfilter utility terminates.

The **[OutputMacros]** section is required and should, at a minimum, define macros for initialization ("Init"), default output ("Default"), carriage return ('CR'), move left ('ML'), ring bell ('BEL' and 'RB'), clear screen ('CS'), align output ('#ALIGN'), horizontal positioning ('@#'), and cursor positioning ('@#,#').

[OutputMacros]

```
; Set the standard driver variables C (COLS) to 80 and R (LINES) to 24
Init=%{80}%SC%{24}%SR%Mi
```

```
; Default output macro used by all character that aren't define elsewhere.  Push the original; (untranslated) character onto stack, output the corresponding character and then call macro "a" to increment the ; column counter by one.
Default=%@%c%Ma
; Characters output with no parameters
'NUL'-'BEL', 'HT'-'VT', 'SO'-'US', 'DEL'= %@%c
; Backspace and, using the "b" macro, decrement the column counter
'BS'=^H%Mb
; Formfeed and, via the "e" macro, clear the column counter
'FF'=^L%Me
; Perform a carriage return, line feed, and clear the column counter
'CR', 'NEL', 'LINESEP'=^M^J%Me
; Translate special space character to a space and increment the column counter
'ENQUAD'-'HAIRSP'=\s%Ma
'PARASEP'=^M^J^J%Me
'U+2190'=\E[27g%Ma
'CS'=\E[0m\E[61\E[2J\E[H%Me
'BP'=\E[1o%{7}%Ms
'EP'=\E[0o%{15}%Ms
'BD'=%{7}%Ms
'ED'=%{15}%Ms
'BH'=\022400%Ma
'XX'=%Mi\E[2J\E[H%Me
; Characters output with one parameter
'#CR'=^M%p1%r^J%;%Me
'#LF'=\E[%p1%dB
; Using the parameter as a repeat count, output 'PI' in loop
'#PI'=%p1%r_%;%Mc
; Define horizontal cursor positioning
@#=%p1%GX%Mm%SX\E[%GX%{1}%+%d'
'#ALIGN'=%p1%GX%p1%m%-%GX%+%SX\E[%GX%{1}%+%d'
; Characters output with two parameters
; Define x,y cursor positioning
@#,#=%p1%SX\E[%i%p2%d;%p1%dH
```

[Macros] Section

The **[Macros]** section of the **TDF** allows the definition of generic macro subroutines which can be referenced from the **[OutputMacros]** section. The left side of the equation specifies the macro name; the right side gives the translation string to be executed when invoked by the caller with a %M*x* statement. Macros are named with a single lower-case letter a **-** z. Just as in the **[OutputMacros]** section, the translation string is a Unicode string with embedded translation statements. A typical use of a macro would be incrementing of the column counter variable "X". Such a macro subroutine reduces the amount of such commands in the main **[OutputMacros]** section. The "a" macro in the example below performs just this function.

Note: The **[Macros]** section, if present, must physically precede the **[OutputMacros]** section in the description file.

```
[Macros]
; Initialization string
i=\E[=15F\E[=0G\E[=15J\E[=OK\E[10m\E[0m\E[0o\E[61
; Increment column counter “X”
a=%GX%{1}%+%SX
; Decrement column counter “X”
b=%GX%{1}%-%SX
; Add pop() to column counter “X”
```

```
c=%GX%p1%+%SX
; Subtract pop() from column counter “X”
d=%GX%p1%-%SX
; Clear column counter “X”
e=%{0}%SX
; Max function, push(pop() max pop())
m=%Sx%Sy%?%Gx%Gy%<%t%Gy%e%Gx%;
; Set foreground color to pop()
s=%Sx%Gx%Gx\E[=%dF\E[=%dJ
```

[OutputUnicodeMapping] Section

The **[OutputUnicodeMapping]** section of the **TDF** controls the final mapping of Unicode characters into the character set or sets compatible with the display terminal. This mapping occurs after translation by the **[OutputMacros] section**. The left side of the equation specifies a range of Unicode characters; the right side begins with the equivalent starting value for that range in the terminal’s character set. The right side consists of up to two additional values, separated by commas. The first is a character set ID, which defaults to 0 if not given. The second, if present, is the letter used to indicate that this character is not LISTable; i.e. should display in “\ooo\” octal notation when used in a BASIC program string.

The Set*n* lines define how to switch the terminal among its available character sets. Character sets are numbered in decimal starting at zero. The terminal driver assumes that Set 0 is active after terminal initialization, and then outputs the appropriate Set*n* command to switch the terminal, when necessary, by keeping track of the active character set.

The [OutputUnicodeMapping] section is required and should, at a minimum, define mapping for the Unicode characters between 0 and 0x7f (the ASCII character set).

```
[OutputUnicodeMapping]
Set0=\E[10m
Set1=\E[11m
0x0000-0x001f=0x00,,n
0x0020-0x007e=0x20
0x007f-0x007f=0x7f,,n
0x00a0-0x00a0=0xff
0x00a1-0x00a1=0xad
0x00a2-0x00a3=0x9b
0x00a5-0x00a5=0x9d
0x00a7-0x00a7=0x15,1
```

[FunctionKeys] Section

The **[FunctionKeys]** section of the **TDF** controls the translation of received multi-character sequences into single 16-bit Unicode characters. The left side of the equation is the Unicode character; the right side gives the equivalent multi-character string. Note that the translation in this case is effectively occurring from right to left in the equation; i.e. the right side of the equation represents terminal characters, the left side is the equivalent Unicode character. To be treated as part of a multi-character sequence, the input characters must match a defined sequence and be received within the **FunctionKeyTimeout** period (see the **[Settings]** section). Any input character that is not translated by this section is passed to the **[InputUnicodeMapping]** section.

```
[FunctionKeys]
’MU’=\E[A
’MD’=\E[B
’MR’=\E[C
’ML’=\E[D
```

```
'END'=\E[F
'MH'=\E[H
'NEXTPAGE'=\E[G
'PREVPAGE'=\E[I
'IC'=\E[L
'F1'=\E[M
'F2'=\E[N
'F3'=\E[O
'F4'=\E[P
```

[InputUnicodeMapping] Section

The **[InputUnicodeMapping]** section of the **TDF** controls the initial mapping of received characters from the terminal into Unicode characters. The left side of the equation specifies a range of Unicode characters; the right side gives the equivalent starting value for that range in the terminal's character set. Note that the translation in this case is effectively occurring from right to left in the equation; i.e. the right side of the equation represents terminal characters, the left side Unicode characters.
The following simple example basically describes a 7-bit US-ASCII terminal, where the high-order bit of incoming characters is ignored.

```
[InputUnicodeMapping]
0x0000-0x007f=0x00
0x0000-0x007f=0x80
```

[InputActions] Section

The **[InputActions]** section of the **TDF** defines the special action, if any, to be performed as specific characters are entered as input. Characters not mentioned in this section are accepted as normal data. Input characters that are used as the leading characters of multi-character sequences in the [FunctionKeys] section should not be as Abort or Escape characters. The left side of the equation specifies the Unicode character; the right side gives the associated action to be performed. Actions are represented by a keyword from the following list:

Keyword	Type of action	Description
`Abort`	Event	Queues an "abort" event, which is used to unconditionally stop a running BASIC program.
`Back`	Editing	Move input position back 1 character.
`Backspace`	Editing	Delete the preceding input character.
`Begin`[1]	Editing	Cursor is moved to the first character of the current input line ("Home" action) and set to input mode. Three options follow; 1 - 'Enter' key line is left unchanged, 2 – A data character replaces the current line with the data character, 3 - an edit character allows editing to the current line. This action should be associated with the 'Begin' character.
`Cancel`	Editing	Clear input buffer.
`Data`	Data	Accept as data and echo (default).
`DataCR`	Data	Translate to 'CR' as a data character and not as a terminator.
`Delete`	Editing	Delete the current input character.
`DeleteEOL`[1]	Editing	Deletes all characters from the current cursor position to the end of line.

[1] dL4 version 2.3 and greater

`EchoSpace`	Data	Accept as data, and echo a SPACE.
`End`	Editing	Move input position to end-of-input.
`Enter`	Editing	Terminate input.
`Escape`	Event	Queues an “Escape” event, which is used to produce an error 99 in BASIC, etc.
`Forward`	Editing	Move input position forward 1 character.
`NextWord`	Editing	Move input position forward 1 word.
`Home`	Editing	Move input position to beginning-of-input.
`Ignore`	Data	Discard the character.
`Illegal`	Data	Discard the character and echo a BELL.
`Insert`	Editing	Toggle input insertion mode on/off.
`Interrupt`	Event	Queues an “Interrupt” event, which is used to trigger an INTSET branch in BASIC.
`NoEcho`	Data	Accept as data, but echo nothing.
`PrevWord`	Editing	Move input position back 1 word.
`Refresh`	Event	Redraws the screen when using Dynamic Windows
`Signal`	Event	Queues a "SignalSelf" event, which usually causes an empty message to be sent to the current port.
`Swap`	Event	Queues a “Swap” event, which usually invokes a pre-determined BASIC program.
`ToggleEcho`	Editing	Toggle echo mode on/off.
`ToggleEchoData`[1]	Editing	Toggle echo with ":" if entering non-echo mode. Returns the toggle character as input data.

```
[InputActions]
’NUL’,’BEL’,’LF’-’FF’,’SO’-’ETB’,’EM’,’GS’-’US’=Illegal
’SOH’=PrevWord
’STX’=Signal
’ETX’=Interrupt
’EOT’=Escape
’ENQ’=ToggleEcho
’ACK’=NextWord
’BS’=Backspace
’CR’=Enter
’CAN’=Cancel
’SUB’=DataCR
’FS’=Abort
’MR’=Forward
’ML’=Back
’IC’=Insert
’DEL’=Delete
’MH’=Home
’END’=End
’F1’=Swap
```

[Settings] Section

The **[Settings]** section of the **TDF** contains miscellaneous terminal and driver parameters. The left side of the equation is the parameter name; the right side defines the parameter value. All of the parameters are optional and will assume a default value if not specified. The currently defined parameters are:

AcceptPartialKey=<boolean>

[1] dL4 2.3 or Greater

If only part of a multi-character key defined in the **[FunctionKeys]** section is received, the default action of the terminal driver is ignore the characters. By setting this parameter to TRUE, the driver will return the received characters as separate input characters.

Background=<RGB>
This specifies the default dL4 window background color.

DL4Term=<boolean>
If true, the parameter enables special dL4Term input and output processing. This option should only be used with dL4Term compatible terminals.

Foreground=<RGB>
This specifies the default dL4 window foreground color. This is the color used for text and graphics.

FunctionKeyTimeout=<number>
This is a numeric value representing the maximum amount of time in seconds allowed between two characters for both to be considered part of the same multi-character (i.e. function key) sequence. The default value is 0.4 seconds.

ProtectedNotDim=<boolean>
In a dL4 window, protected characters drawn after a ‘BP’ mnemonic are normally displayed in “dimmed” mode. If ProtectedNotDim is set to TRUE, protected characters are displayed like any other characters using the current attributes and font color. The default value of ProtectedNotDim is FALSE.

StartUpInWindow=<boolean>
This specifies to the terminal driver that the initial screen, which is usually the physical terminal screen, should instead be a dL4 window overlaying the entire screen. This is similar to executing a "Window On" in a dL4 program. This option is used to emulate protected characters on a terminal that does not support protected characters. The default value is FALSE.

TabBySpacing=<boolean>
In a dL4 window, the TAB function normally skips directly to the new output position without changing any characters. With TabBySpacing set to TRUE, the TAB function will move to the new position by outputting spaces. The default value is FALSE.

Boolean values ("<boolean>") have values of true ("t", "true", "on", "y", "yes", or "1") or false ("f", "false", "off", "n", "no", or "0"). RGB color values ("<RGB>") are 24-bit integers that can be specified in decimal ("16777215"), octal ("077777777"), or hexadecimal ("0xffffff"). Since RGB values consist of three eight bit components for the red, green, and blue color intensities, the hexadecimal form may be most convenient. For example, the value 0x804000 combines a red value of 128 ("80") and a green value of 64 ("40") to form the color brown.

```
[Settings]
FunctionKeyTimeout=.4

StartUpInWindow=False

TabBySpacing=True

; Set default window colors to white on black

Foreground=0xffffff

Background=0
```

Section Ordering

The sections of a terminal description file may be arranged in any order. However, there is an optimal ordering which is most efficient for loading and processing the file. The order is:
[OutputUnicodeMapping]
[Macros]
[OutputMacros]
[InputUnicodeMapping]
[FunctionKeys]
[InputActions]
[Settings]