NIAKWA PROGRAMMING LANGUAGE

INTEL UNIX SUPPLEMENT

1st Edition - May 1994

Niakwa, Inc.
23600 N. Milwaukee Avenue
Mundelein, IL 60060

PHONE: (708) 634-8700; FAX (708) 634-8718; TELEX 3719965 NIAK UB

PREFACE

P.1 The NPL Intel UNIX Supplement

The Niakwa Programming Language (NPL) Intel UNIX Supplement is designed as an aid to the installation, operation and operating systems-specific features of the NPL Interpreter, Compiler, RunTime program and related utilities in the following environments:

Hardware	Operating Systems	Notes
IBM-Compatible PC (80386 or greater)	Altos UNIX System V/386 rev. 3.2 or greater	32-bit implementation of NPL
IBM-Compatible PC (80386 or greater)	AT&T UNIX System V/386 rev. 3.2 or greater	32-bit implementation of NPL NCR System 3000 Wyse 5000, 7000, and 9000
IBM-Compatible PC (80386 or greater)	Interactive UNIX 386/ix rev. 2.0.2 or greater	32-bit implementation of NPL
IBM-Compatible PC (80386 or greater)	SCO UNIX System V/386 rev. 3.2 or greater	32-bit implementation of NPL
IBM-Compatible PC (80286 or greater)	SCO Xenix System V/286 rev. 2.x or greater	16-bit implementation of NPL

P.1.1 Prerequisite Knowledge

This guide assumes at least a basic knowledge of the supported hardware and operating system.

P.1.2 How to Use the NPL Documentation

The NPL documentation has been structured into generic and operating environment dependent manuals to allow developers easy reference to detailed platform-specific information. The NPL Technical Reference Guide (TRG) consists of two manuals, the NPL Programmer's Guide and the NPL Statements Guide. The TRG set is intended as a generic guide for programmers in the correct use of NPL and its program development and debugging facilities on all supported environments.

The TRG set is designed to be used in conjunction with the NPL operating system-specific supplements (that clearly document the operating system-dependent features of NPL on that specific operating system). Some supplements also contain a series of operating environment-specific addenda. These addenda discuss the NPL operation and options that are either unique to a given operating environment, or function differently from platform to platform.

Additionally, each supplement contains a set of Release Notes, providing information pertaining to an operating system-specific supplement that was not available at press time.

Refer to the Preface in the NPL Programmer's Guide for more information on the use of the NPL documentation.

P.1.3 How to Use this Supplement

The NPL Intel UNIX Supplement discusses the operation of NPL under Intel UNIX based operating environments (SCO Xenix is also supported as a 16-bit product).

P.1.4 Notational Conventions

This guide uses the following notational conventions:

NOTE: Notes provide information of particular importance.

WARNING--Warnings are special conditions that require extra care by the user.

HINT: Hints provide helpful comments pertaining to the use of particular features.

The following conventions are used in the illustration and definition of NPL:

- Each statement appears on a separate page, with the statement as a page header.
- The general form of each statement is enclosed within a box.
- Uppercase letters ("A" through "Z"), digits ("0" to "9"), and special characters (such as "$", "#", ":") must always appear exactly as presented in the general format.
- All lower-case words indicate information that the user must supply. These words appear in *italic* type.

  For example:

  ```
  LEN (alpha-variable)
  ```

  The user must supply the alpha-variable.

- When braces, "{ }", enclose a vertically stacked list, or a horizontal list with each item separated by a comma (","), the user must choose one of the options within braces. Information within braces is shown in *italic* type.

  For example:

  ```
  ALL ({literal-string, alpha-variable, two-hexdigits})
  ```

  or

  ```
  ALL   ({literal-string  })
         {alpha-variable  }
         {two-hexdigits   }
  ```

Here, the ALL instruction must be followed by one and only one of the items in the list.

- Brackets, "[ ]", indicate that the enclosed items are optional. When brackets enclose a vertical list or a horizontal list, the user may specify one or none of the items. Information within brackets is shown in *italic* type.

  For example:

  ```
  INPUT [literal-string,] variable [,variable]...
  ```

  Here, the INPUT instruction may optionally contain a literal-string followed by an optional comma preceding the required "variable". Additional variables may optionally be specified.

  or:

  ```
  CLEAR [V                                  ]
        [N                                  ]
        [P [line-number1][,[line-number2]]]
  ```

  Here, either the V, N, or P parameter may be specified, but no parameter is required. Line-number parameters may be optionally specified only if the "P" parameter is specified.

- The presence of an ellipsis (...) within any format indicates that the unit immediately preceding the ellipsis can occur one or more times in succession.

  For example:

  ```
  DEFFN'integer[(variable[,variable]...)]
  ```

  Here, any number of "variable" may be specified, but the format ",variable" must be used for the second and subsequent "variables".

- All other punctuation such as commas or parentheses must be included where shown.

P.1.5 Terminology

Throughout this Supplement, the following terms are used:

Niakwa NPL Development Package
This refers to the entire contents of the package, including the NPL Development Package diskettes, the NPL Technical Reference Guide (Programmer's and Statements Guides), and the NPL Supplement. This term may frequently refer to just the software elements of this package.

Niakwa NPL Technical Reference Guide
This refers to the NPL Statements Guide and Programmer's Guide.

NPL Compiler
This refers to the actual compiler program itself, b2c.

Niakwa NPL Development Package Diskettes
This refers to the physical diskettes delivered as part of the Niakwa NPL Development Package. These diskettes contain the contain the NPL Compiler, Utilities, etc. software used in NPL software development.

Niakwa NPL RunTime Package
This refers to the RunTime Installation Guide and diskettes. This package is used by the Authorized NPL Developer for executing and testing the application object code generated by the compiler. It is also used by end-users for executing application object code purchased from a Authorized NPL Developer.

Niakwa NPL RunTime Program
This refers to the Interpretive (rti) or Non-Interpretive (rtp) RunTime programs.

Niakwa NPL RunTime Program Diskette(s)
This refers to the physical diskette(s) delivered as part of the Niakwa NPL RunTime Package. The first diskette of this package is also referred to as the Gold Key diskette.

TABLE OF CONTENTS

CHAPTER 1

INTRODUCTION

1.1 Overview

The Niakwa Programming Language Supplement for Intel UNIX is intended as an aid in the correct installation and use of the NPL Interpreter, Compiler and RunTime Program under Intel UNIX.

Section 1.2 describes the contents of the NPL Development Package.

Section 1.3 describes the contents of the NPL RunTime Package.

Section 1.4 discusses the main features of the RunTime Program which are specific to Intel UNIX.

Section 1.5 provides an overview of the NPL Intel UNIX Supplement.

1.2 Contents of the Niakwa Development Package

The NPL Development Package is intended for software vendor use in the development and execution of application software under Intel UNIX. The NPL Development Package consists of the NPL Intel UNIX Supplement and 7 diskettes.

NOTE: The NPL Development Package for Intel UNIX is provided on both 5-1/4" high density 1.2MB and 3-1/2" low density 720K diskettes. In addition, both the 286 (Xenix) and 386 (UNIX) version are provided.

The following is a description of the contents of these diskettes.

Compiler Diskette

b2c	The NPL Compiler program.
b2cx	Example script files showing command-line use of the compiler designed to simplify use.
ENABLED	The "enable" file necessary for allowing the Interpretive RunTime program to function.
EXSNPL.BS2	A diskimage containing NPL Release IV example programs.
NPLEXAM.BS2	A diskimage containing the NPL Release IV example programs as discussed in Chapter 17 of the NPL Programmer's Guide.
NPLSYS.BS2	A diskimage containing NPL Release IV library modules used by the example programs as discussed in Chapter 3 of the NPL Statements Guide.
B2CHELP.HLP	A text file containing help entries for the NPL compiler.
B2CHELP.IDX	A file containing the index used when the NPL compiler help file is accessed.
BOOT.OBJ	The default RunTime boot program in compiled p-code format.

GOLDKEY.OBJ	A boot program that converts a serial number into its corresponding #GOLDKEY value.
REL4EXAM.OBJ	A boot program used to start the Release IV example programs.
BOOT.SRC	The default RunTime boot program in source format.

Utilities Diskette

util	Script file that starts the NPL Utilities..
utilinst	Text displayed containing start-up menu instructions for the NPL Utilities when util script is run.
UTILITY.BS2	A diskimage file containing the NPL Utilities.
UTILHELP.HLP	An indexed help files for use by the NPL Utilities.
UTILHELP.IDX	A file containing index listings used when the help file is accessed.
UTILITY.OBJ	A compiled version of the boot program to start up the NPL Utilities.
UTILITY.SRC	A source code version of the boot program for the Utilities.

Terminal Support Files Diskette

w2236	Source for termcap entry (used on Alto UNIX).
wangmode	Turns on the Wang 2x36 flow control for Altos UNIX.
VTFONT.AL5	Font file used for downloading the NPL screen character set for the Altos V terminals.
IBMFONT0.EGA	A font file used for downloading the screen character set for EGA cards on an IBM compatible PC.
ttys.exp	An example of an NPL ttys file.

IBMFONT0.HGA	A font file used for downloading the screen character set for HGA cards on an IBM compatible PC.
WYKEYS.OFF	File necessary for correct keyboard operation in application-key mode of Wyse 150 and 160 terminals in Wyse 60 emulation.
WYKEYS.ON	File necessary for correct keyboard operation in application-key mode of Wyse 150 and 160 terminals in Wyse 60 emulation.
FONT.TBL	A character font file which can be used as a reference font file by the NPL Font Table Editor.
FONTIBME.TBL	Modifiable font files used to create downloadable font file for the EGA controller.
FONTIBMH.TBL	Modifiable font files used to create downloadable font file for the IBM EGA board and Hercules MGA board.
FONTxxxx.TBL	Font table files used for editing and creating the above downloadable font files.
VTFONT.VT200	A font file used for downloading the screen character set for the VT200 series terminals.
VTKEYS.VT200	A file necessary for downloading the uppercase key definitions for VT200 series terminals.
WYKEYS.WY370	A file necessary for proper keyboard operation of a Wyse 370 terminal.
WYFONT.WY60	A font file used for downloading the screen character set for the Wyse 60 terminals.
KEYBOARD.xxx	A series of keyboard translation tables used for handling differences between the keyboard and hex codes expected by NPL programs. These files are used on the various NPL-supported terminals that can be used for remote maintenance with the Redirect feature of the RunTime.

SCREEN.xxx	A series of screen translation tables used for displaying the NPL standard character set. These files are used on the various NPL-supported terminals that can be used for remote maintenance with the redirect feature of the RunTime.
W370FONT.xxx	Font files used for downloading the screen character set (80 or 132 columns) for the Wyse 370 terminals.

BESDK Diskette

This diskette contains the required files necessary to allow NPL to interface with external subroutines written in other languages (one diskette for SCO Xenix or two diskettes for Intel UNIX). For a complete description of all files on this diskette, refer to Chapter 11.

1.3 Contents of the Niakwa NPL Runtime Package

The Niakwa NPL RunTime Package is intended for:

- Vendor use in executing and testing application object code developed in NPL.
- End-user use for execution of application object code purchased from an Authorized NPL Developer.

The NPL RunTime Package is packaged as two unique products. The first product is the Niakwa RunTime Package and is normally shipped for new orders of NPL. The second product is the Niakwa Upgrade RunTime Package. This may be ordered for upgrades to existing customers in cases when the customer already has a Release III RunTime installed on the system. The advantage of the Upgrade RunTime is that the old Gold Key does not need to be returned. The contents of each product are different. Refer to the other sections in this chapter for more details on the contents of the Niakwa RunTime and the Niakwa Upgrade RunTime Packages.

The Niakwa RunTime and Niakwa Upgrade RunTime Packages are available on both 5-1/4" and 3-1/2" media. The files for each RunTime Package are contained on one 5-1/4" or 3-1/2" diskette.

1.3.1 Niakwa RunTime Package Files

The Niakwa RunTime Package (for new installations) physically consists of the Niakwa RunTime Package Installation Guide and one 5-1/4" or 3-1/2" diskette. These diskettes are labeled as follows:

- Niakwa NPL RunTime Package - GOLD KEY

The "Gold Key" is necessary to complete any task requesting or requiring the Gold Key Diskette (i.e., installing the Gold Key security to the hard drive, passing security, etc.).

Niakwa NPL RunTime Package File Listing

The Niakwa RunTime Package contains the following files:

clearmem	UNIX RunTime only. Program that clears the shared memory (without rebooting the system) if the old Release III RunTime programs were run before the Release IV versions.
commonmem	Xenix RunTime only. A file used internally by the RunTime.
hexedit	An utility program that is used by Niakwa to help diagnose NPL security problems.
hexdump	An utility program that is used by Niakwa to help diagnose NPL security problems.
rti	The Interpretive RunTime program.
rtp	The Non-interpretive RunTime program.
userfix	UNIX RunTime only. A utility that allows the developer to reset the user count in situations where the user count is greater than the actual number of NPL users on the systems.
xeninst	A shell script used for installing or recalling the NPL copy protection on the hard drive.
NIAKSER.DAT	File used by the Runtime security check.
install.err	Installation program error messages.

ERRORMSG.HLP	A text file that contains the error messages that are displayed when using the Interpretive RunTime program.
RTIXERR.HLP	A text file that contains the UNIX error messages that are displayed optionally when using the Interpretive RunTime program.
ERRORMSG.IDX	File containing the index used when the ERRORMSG.HLP file is accessed.
RTIXERR.IDX	File containing the index used when the RTIXERR.HLP file is accessed.
setup.nodes	A shell script file necessary for establishing special device names required by the NPL RunTime.
XENINST.OBJ	File used by the install procedure.

1.3.2 Niakwa NPL Upgrade RunTime Package

The Niakwa Upgrade RunTime Package physically consists of the Niakwa RunTime Package Upgrade RunTime Installation Guide and one 5-1/4" or 3-1/2" diskette. These diskettes are labeled as follows:

- Niakwa NPL Upgrade RunTime Package

The 5-1/4" or 3-1/2" diskette labeled Niakwa Upgrade RunTime Package is necessary to complete the upgrade procedure.

Niakwa NPL Upgrade RunTime Package File Listing

The Niakwa Upgrade RunTime Package contains the following files:

RTIUPG	The Interpretive RunTime program.
RTPUPG	The Non-Interpretive RunTime program.
upgrade	A file used to start the upgrade procedure.
UPGRADE.BS2	A diskimage used by the upgrade procedure.

ERRORMSG.HLP	A text file that contains the error messages that are displayed when using the Interpretive RunTime program.
RTIXERR.HLP	A text file that contains the UNIX error messages that are displayed optionally when using the Interpretive RunTime program.
ERRORMSG.IDX	File containing the index used when the ERRORMSG.HLP file is accessed.
RTIXERR.IDX	File containing the index used when the RTIXERR.HLP file is accessed.
UPGR0.OBJ	A file used by the upgrade procedure.

The NPL Upgrade RunTime Package is ordered separately from Niakwa for each end-user who is upgrading from a Release III UNIX or Xenix RunTime Package. Refer to Chapter 2 for further details on the Niakwa NPL Upgrade RunTime Package.

1.4 UNIX-Specific Features of the NPL Runtime

The RunTime program for Intel UNIX supports multi-user operation on individual terminals connected to the UNIX environment. In addition to the standard NPL Release IV features, specific features under Intel UNIX include:

- The ability to individually identify workstations or terminals on the system under program control.
- Support for variables larger than 64K (UNIX only). Refer to Chapter 4 of the NPL Programmer's Guide.
- Support of mixed language programming with call-back capabilities to NPL FUNCTIONs or PROCEDUREs.
- Support of 2.88MB "raw" diskettes on systems that support these formats. Refer to Section 5.2.

1.5 Intel UNIX Supplement Overview

The Intel UNIX Supplement is designed to provide information on operating NPL on Intel UNIX based systems. This supplement documents platform-specific features only. General language features are discussed in the NPL Programmer's Guide or the NPL Statements Guide.

Chapter 2 of the Supplement discusses installation and configuration requirements of NPL on Intel UNIX systems. It also discusses the NPL Gold Key security and security installation.

Chapter 3 discusses configuration of the NPL system including location of files, and use of auxiliary files.

Chapter 4 discusses RunTime operations, describing various command-line startup options available, and exiting the RunTime.

Chapter 5 discusses devices supported by NPL. These include diskimage files, "raw" diskettes, monitors/controllers, printers, tape drives, serial devices, math coprocessors, and mice.

Chapter 6 discusses terminal characteristics under Intel UNIX including operating system and supported terminal characteristics. In addition, the NPL support of the system console is also discussed.

Chapter 7 discusses multi-user issues for Intel UNIX systems and how they can co-exist with networks.

Chapter 8 discusses language features that are specific to Intel UNIX, including differences with the system variables $MACHINE and $OPTIONS.

Chapter 9 covers operation of the B2C compiler and conventions used under Intel UNIX.

Chapter 10 discusses options for porting programs and data to Intel UNIX systems from other platforms.

Chapter 11 discusses the use of mixed-language programming under Intel UNIX to create external libraries.

Appendix A contains some common problems and solutions encountered while configuring or executing NPL under Intel UNIX.

Appendix B contains common Intel UNIX error codes encountered during disk I/O or native operating system commands.

Appendix C discusses performance issues with NPL under Intel UNIX.

Appendix D diagrams "raw" device compatibility for all current NPL-supported systems.

CHAPTER 2

INSTALLATION

2.1 Overview

This chapter provides instructions for installing the NPL Development and RunTime Packages under Intel UNIX.

Section 2.2 discusses the operating system and hardware requirements for use of the NPL.

Section 2.3 discusses the operating system configuration requirements for the use of NPL.

Section 2.4 discusses installing the NPL Development Package.

Section 2.5 discusses the NPL RunTime's Gold Key security.

Section 2.6 discusses the installation of the NPL RunTime Package under Intel UNIX.

Section 2.7 discusses the installation of all Intel UNIX based NPL Upgrade RunTime Packages.

2.2 OS and Hardware Requirements

The following section discusses the operating system and hardware requirements for the Intel UNIX implementation of NPL.

2.2.1 System Requirements

The Intel UNIX implementation of NPL is designed to operate on systems meeting the following requirements:

Hardware	**Operating Systems**	**Notes**
IBM-Compatible PC (80386 or greater)	Altos UNIX System V/386 rev. 3.2 or greater	32-bit implementation of NPL
IBM-Compatible PC (80386 or greater)	AT&T UNIX System V/386 rev. 3.2 or greater	32-bit implementation of NPL NCR System 3000 Wyse 5000, 7000, and 9000
IBM-Compatible PC (80386 or greater)	Interactive UNIX 386/ix rev. 2.0.2 or greater	32-bit implementation of NPL
IBM-Compatible PC (80386 or greater)	SCO UNIX System V/386 rev. 3.2 or greater	32-bit implementation of NPL
IBM-Compatible PC (80286 or greater)	SCO Xenix System V/286 rev. 2.x or greater	16-bit implementation of NPL

NOTE: Refer to Niakwa's BBS or your authorized sales representative for current information on newly approved/tested hardware and operating system versions.

Each of the NPL RunTime programs requires a different amount of memory. The following chart indicates the approximate memory requirements of each and the per user overhead as well.

	Interpretive RunTime (rti)	**Non-Interpretive RunTime (rtp)**
Base RunTime (sharable)	254K (UNIX) 217K (XENIX	144K (UNIX) 133K (Xenix)
Per user overhead (non-sharable)	203K (UNIX) 216K (Xenix)	198K (UNIX) 214K (Xenix)

2.2.2 Supported Terminals

Supported NPL terminals include:

- Altos III and V
- Bull HDS 1 and 3
- DEC VT100 and VT200 series
- IBM 3151
- NCR 4970
- Spectrix SPX 701
- Wang 2110A
- Wang 2236DE, and 2236DW
- Wyse 50, 60,150, 160, and 370

Refer to Chapter 6 or Appendix D of the NPL Programmer's Guide or for a complete discussion of all Niakwa-supported terminals.

2.2.3 Printer Support

Both parallel and serial system printers are supported. In addition, local printers are supported for terminals with parallel or serial printer ports. Refer to Section 5.5 for additional information on printer support under UNIX.

2.3 NPL Configuration Requirements

The following represent configuration requirements for NPL under Intel UNIX.

2.3.1 System Parameters

The minimum system parameters for configuring the host operating system are as follows:

Inodes	150
Open files	150
Max processes	100

NOTE: The above settings are typically sufficient for up to 16 concurrent NPL users. Application specific considerations or more users may require these settings to be set to higher values.

2.3.2 Modifying the .profile File

Each UNIX operating system contains the file /etc/profile which contains global login information for all users. In addition, each user has his own .profile file which is located in the user's home directory. These files all contain operational parameters and can be used to execute shell script files and initialize environment variables. The /etc/profile file is used to setup system wide parameters and shell scripts. The user's specific .profile file is used to setup parameters and shell scripts specific to the individual user. Refer to the appropriate UNIX reference manual for further information on the /etc/profile and .profile files and setting up users.

These profile files are also used to perform several functions for the operation of NPL. These include:

- To set up the alternate path to the /usr/BASIC2C directory.
- To set up the default file protection status for any new files created by the user.
- To automatically load and execute a specified shell script file.
- Properly set the terminal type (BASIC2C_TERM--refer to Section 6.3.1 for valid entries). In addition, for some terminals, other commands may be required. Refer to Chapter 6 for details.
- Optionally set the BASIC2C_ID system variable. BASIC2C_ID can be used to generate unique #ID values. Refer to Chapter 7 for details.

For convenient operation of NPL applications, certain parameters in the /etc/profile and in the appropriate default user .profile files should be modified before any user is "created" on the system.

The following changes should be made to the /etc/profile to set up /usr/BASIC2C (or whatever directory the NPL files are to be located in) as an alternate path.

NOTE: These changes should be performed while logged in as super-user (root) and from an UNIX text editor (e.g., vi, ed). It is a good idea to make a backup of the file before editing.

1. Modify the "PATH=" line by adding to the end:

```
:/usr/BASIC2C:.
```

NOTE: The dot (.) at the end of the PATH statement must be included. This indicates the current directory should be included as an alternate path. This causes UNIX to search the current directory for commands.

For example:

Modify the "PATH=" line in the /etc/profile file from:

```
PATH=:/etc:/bin:/usr/bin
```

to:

```
PATH=:/etc:/bin:/usr/bin:/usr/BASIC2C:.
```

NOTE: It is recommended to make the following changes to the .profile file for each specific user or to the default user .profile to set the default file protection status, and optionally specify a program to be executed at login time.

WARNING--Each flavor of UNIX has its own way of configuring users. Refer to the UNIX administration documentation for more information.

2. Set BASIC2C_TERM (refer to Chapter 6) and optionally BASIC2C_ID (refer to Chapter 8) system variables and execute any terminal specific commands required.

3. Optionally, the NIAKWA_RUNTIME environment variable setting. Refer to Section 2.5.4.

4. Optionally, a UNIX command to be automatically executed at login time may be added to the .profile file. If this is done, this should be added as the last line in the file.

   For example, .profile entries may include:

   ```
   umask 022
   cd /usr/BASIC2C
   rti UTILITY
   ```

   The above lines when added to a users .profile file performs the following tasks:

   - Sets for other users, read only access to any file created by this user.
   - Changes the directory to /usr/BASIC2C.
   - Executes the RunTime's UTILITY program.

   To allow these changes to take affect, Logoff (by pressing Ctrl-D or entering "lo-gout") and login again.

2.4 Installing the NPL Development Package

This section discusses the installation of the NPL Development Package under Intel UNIX. The NPL Development Package consists of seven diskettes labeled as follows:

- Compiler Diskette (one each for UNIX and Xenix)
- Terminal Support Diskette (supports both UNIX and Xenix)
- Utilities Diskette (supports both UNIX and Xenix)
- BESDK Diskette (two for UNIX and one for Xenix)

Each diskette is provided on a 3-1/2" 720K and a 5-1/4" 1.2MB diskette.The contents of these diskettes are described in Section 1.2. The first three diskettes are installed in the /usr/BASIC2C directory. The BESDK diskette has a separate installation process. Refer to Chapter 11 for details.

2.4.1 Intel UNIX Diskette Drives

Under UNIX, diskette drives are accessed by using a special device name. Each device name refers to a specific diskette drive and the media expected to be in the drive. Intel UNIX refers to each drive as drive 0 or drive 1.

The following table illustrates several examples on accessing Intel UNIX floppy drive with different media. Keep in mind that different flavor of UNIX may use other designations as well. Refer to the appropriate UNIX documentation for more information.

Drive	3-1/2" 720K	5-1/4" 1.2MB
0	/dev/fd0135ds9*	/dev/fd096ds15
1	/dev/fd1135ds9*	/dev/fd196ds15

* Prior to running setup.nodes, Interactive and AT&T flavors of UNIX require that /dev/dsk/f0q9dt 720 or /dev/dsk/f1q9dt 720 be used.

NOTE: The designation of drive "0" is used in this manual for the purpose of documentation.

2.4.2 Installing the Development Software

The Development Package diskettes have been produced in UNIX Tape Archive ("tar") format on 3-1/2" 720K or 5-1/4" 1.2MB media. In this form, the diskettes may be easily copied to the system's hard drive by using the UNIX tar command.

NOTE: UNIX is case sensitive in assigning file names and directory names. All references to file names, directory names, and UNIX commands in this document accurately reflect the appropriate case.

When copied to the system's hard drive, all NPL software is placed in the /usr/BASIC2C directory. The /usr/BASIC2C directory is created automatically as part of the tar procedure. As such, it is not necessary to create the /usr/BASIC2C directory before running tar.

Installation Steps

To install the Niakwa NPL Development Package's software from floppy drive 0 to the system's hard drive, follow these steps:

1. Login to UNIX as "root" (the super-user login)

2. Enter the following UNIX commands:

```
cd /           Select the current directory as root.
umask 0000     Sets protection mode to full access for all users.
```

3. For installing from drive 0 enter:

   for 3-1/2" 720K media

```
tar xvfk /dev/fd0135ds9 720
```

NOTE: For Interactive and AT&T UNIX installation use the following until setup.nodes is run:

```
tar xvfk /dev/dsk/f0q9dt 720
```

   for 5-1/4" 1.2MB media

```
tar xvfk /dev/fd096ds15 1200
```

   The screen will display a list of files being copied to the system's hard drive by the tar process. Repeat this command for each of the development diskettes (excluding the BESDK diskette).

NOTE: This procedure is identical for both developer and end user sites. Refer to the UNIX Commands Directory for details on the UNIX tar utility.

2.4.3 Installing the NPL External Subroutine Development Kit (BESDK)

The BESDK diskette has a separate installation procedure. Refer to Chapter 11 for the installation procedure for the BESDK diskette NPL (formally, Basic-2C) External Subroutine Development Kit.

2.5 NPL Runtime Security

The following section discusses the NPL security.

2.5.1 Security Overview

The RunTime programs are protected from unauthorized copying by use of a security scheme. The Gold Key diskette issued by Niakwa contains a physically encoded serial number. The RunTime Package contained on the Gold Key contains the same encrypted serial number.

The protection being used is an "install based security" by which the RunTime program can be copied to a single disk drive. With this type of security the RunTime does not require insertion of the Gold Key diskette. The RunTime program can be "installed" and "recalled" to allow changing systems.

NOTE: Once the RunTime is installed, the Gold Key diskette is used as a backup in case the installed security on the hard drive is lost.

2.5.2 The NIAKSER.DAT File

At start-up time, the NPL RunTime Program attempts to read the NIAKSER.DAT file and extract the encrypted "Serial Number" from it. To find the NIAKSER.DAT file, the RunTime first looks at the current directory, then the directory NIAKWA_RUNTIME is set to (if used), and finally the /usr/BASIC2C directory.

The RunTime fails whenever the NIAKSER.DAT file cannot be read, found, or contains invalid data. If this occurs, the following error message may be displayed:

```
NIAKSER.DAT missing or damaged (error code)
RunTime program canceled.
Contact your distributor for assistance.
```

where the numeric code generated can be one of the following:

Error Code	**Description**
Code 0	Cannot find NIAKSER.DAT.
Code 1	Cannot read NIAKSER.DAT.
Codes 2-5	Invalid data in NIAKSER.DAT. File is probably damaged

In addition, the RunTime generates an error message if the internal revision level in the RunTime is greater than the revision level contained in NIAKSER.DAT.

NOTE: If it is necessary to contact Niakwa's Technical Support regarding one of the above errors, please be sure to note the exact error code received.

2.5.3 The Security Check

Based on the "Serial Number" extracted from NIAKSER.DAT, the NPL RunTime security check is comprised of the following steps:

1. The current directory is checked for the matching security fingerprint protection. If the hard drive protection is found, execution continues.

2. If the fingerprint is not found in the current directory, the directory specified in the NIAKWA_RUNTIME environment variable is checked. If the protection is found, execution continues.

3. If the fingerprint is not found in the directory specified by the NIAKWA_RUNTIME environment variable, the /usr/BASIC2C directory is checked. If the hard drive protection is found, execution continues.

4. If the hard drive protection check fails all of the above checks, normal diskette based security procedures must be performed. Here, the "Mount Gold Key" message appears whether or not the Gold Key diskette is in the drive. When this occurs, insert the original Gold Key diskette in the diskette drive. If the protection is found, execution continues and the Gold Key diskette may be removed and returned to a safe place. The Once-a-Day feature then takes effect for subsequent executions of the RunTime until the system is rebooted or the date changes. Refer to Section 2.5.6 for details on the Once-a-Day Security feature.

5. If the fingerprint is not found on the diskette, the "Mount Gold Key" screen is redisplayed.

Refer to Section 2.5.5 for more information on the security screen.

2.5.4 NIAKWA_RUNTIME Environment Variable

The NPL RunTime supports the use of an environment variable to specify the directory in which the RunTime system files are located. These system files include:

NIAKSER.DAT
ENABLED
Screen and keyboard translation files
Error message files
Patch files

NOTE: For purposes of upward compatibility, the NPL RunTime still looks in /usr/BASIC2C if files are not found in the NIAKWA_RUNTIME directory.

The search path used by the RunTime is as follows:

Current directory
NIAKWA_RUNTIME directory
/usr/BASIC2C directory

To establish the NIAKWA_RUNTIME directory, use the UNIX export command. This is best placed in the user’s .profile file. For example:

```
NIAKWA_RUNTIME=/usr/NPL
export NIAKWA_RUNTIME
```

This instructs the RunTime to look for system files in the /usr/NPL directory.

Locating the NPL Software in a Directory other than /usr/BASIC2C

If the NPL software is to be moved to a different directory other than /usr/BASIC2C (where it is automatically installed), the following must be considered:

- The NIAKWA_RUNTIME Environment Variable must be set to the new directory and exported.
- The setup.nodes script should be executed while the NPL file still reside in /usr/BASIC2C directory (prior to being moved).
- Even if the NPL files are moved to a different directory, the /usr/BASIC2C directory must still be available. When the RunTime executes, it creates a few files that must be placed in the /usr/BASIC2C directory. As such, it is necessary to recreate the /usr/BASIC2C directory once the NPL files have been moved. For example:

```
mv /usr/BASIC2C /usr/NPL
mkdir /usr/BASIC2C
```

NOTE: It is necessary to use the UNIX mv (move) command instead of cp (copy) to move the files to another directory.

2.5.5 The Security Screen

In the event that the Gold Key "Security Fingerprint" is not installed on the system’s hard drive or the hard drive based security check fails, normal diskette based security procedures must be performed.

In this event, the following prompt is displayed:

```
Please mount your GOLD KEY diskette.
```

NOTE: This prompt appears whether or not the diskette is in the drive:

When this occurs, mount the original Gold Key diskette with its matching "Serial Number" in the diskette drive. If the "Serial Number" is not found on the diskette or if the serial number is found but does not match, the security check fails and the message "Please mount your Gold Key diskette" remains on the screen.

NOTE: If the Gold Key is required, it must be the original Gold Key containing the "Security Fingerprint". For example, if a 3.20 RunTime is installed and upgraded to Release IV, the 3.20 Gold Key is required to pass the diskette-based security check (the Upgrade RunTime diskettes contain no security fingerprint that can be used to pass the RunTime security check).

When performed, the security check always takes place at the beginning of the RunTime's execution. Once the security check has been completed, the Gold Key diskette may be removed and returned to a safe place.

2.5.6 Once-a-Day Security

The Once-a-Day security feature is designed to eliminate the need for the Gold Key diskette to be inserted each time the NPL RunTime is executed. The Once-a-Day security feature requires that the first user to execute the RunTime program, after booting the system, be logged in as the super-user. The security check is only issued on the first attempt to use the NPL RunTime for that day. The Once-a-Day security check is functional when the RunTime security is diskette based and takes effect for subsequent executions of the RunTime until the system is rebooted or the date changes.

2.5.7 What to Do if the Gold Key Security is Lost

If the security check fails on the hard drive, the installed Gold Key security may have been lost. The first step is to check that the RunTime program has been installed. This can be done by trying to reinstall the RunTime program. Refer to Appendix A for more details.

2.5.8 Replacement RunTimes

If the Gold Key security becomes lost, a replacement RunTime can be ordered from the Authorized NPL Distributor.

NOTE: **The RunTime can be executed normally, using the Gold Key as a backup until the replacement arrives.**

2.6 Installing the NPL Runtime Package

The following sections discusses the installation the NPL RunTime Package. The NPL RunTime Package consist of a single diskette, which is labelled as follows:

- Niakwa NPL RunTime Package - Gold Key

Refer to Section 1.3 for details on the contents of the NPL RunTime Package for Intel UNIX. When complete, all system software necessary for execution of the NPL RunTime is located in the /usr/BASIC2C directory on the system's hard drive.

NOTE: **The Development Package must be installed to facilitate development of NPL application programs. The installation of the NPL Development Package is discussed in Section 2.4.**

2.6.1 NIAKWA_RUNTIME Environment Variable

The RunTime supports the use of an environment variable to specify the location of the NPL files. Refer to Section 2.5.4 for more information.

2.6.2 Copying the Software to the Hard Disk

The RunTime Package diskettes have been produced in the standard UNIX Tape Archive ("tar") format on either 3-1/2" 720K or 5-1/4" 1.2MB diskettes. In this form, the diskettes may easily be copied to the hard drive by using the tar command.

NOTE: **UNIX is case sensitive in assigning file names and directory names. All references to file names, directory names, and UNIX commands in this document accurately reflect the appropriate case.**

When copied to the hard drive, all NPL software is placed in the /usr/BASIC2C directory. The /usr/BASIC2C directory is created automatically as part of the tar procedure. As such, it is not necessary to create the /usr/BASIC2C subdirectory before running tar.

Installation Steps

To install the Niakwa NPL RunTime Package's software from floppy drive 0 to the system's hard drive, follow these steps:

1. Login to UNIX as "root" (the super-user login).

2. Enter the following UNIX commands to install the NPL RunTime files from drive 0 to the system's hard drive:

```
cd /          Select the current directory as root.
umask 0000    Sets protection mode to full access for all users.
```

3. For installing from drive 0 enter:

   for 3-1/2" 720K media

```
tar xvfk /dev/fd0135ds9 720
```

NOTE: For Interactive and AT&T UNIX installation use the following until setup.nodes is run:

```
tar xvfk /dev/dsk/f0q9dt 720
```

for 5-1/4"" 1.2MB media

```
tar xvfk /dev/fd096ds15 1200
```

The screen will display a list of files being copied to the system's hard drive by the tar process.

2.6.3 Installing the Gold Key Security

Niakwa highly recommends installing the security fingerprint on the hard drive. When the security fingerprint is installed on the hard drive, this avoids checking the diskette for the security, making the RunTime security appear transparent to the end user and minimizing the possibility of the Gold Key being lost or damaged.

NOTE: If the RunTime files were moved to another directory other than /usr/BASIC2C, the NIAKWA_RUNTIME environment variable must be set before the installation of RunTime Security. Refer to Section 2.5.4 for more information.

Installation Steps

The following steps are to be performed for installing the NPL Gold Key security fingerprint to the system's hard drive:

NOTE: Interactive and AT&T UNIX installations must run the setup.nodes script before proceeding with the Gold Key security installation and several device names are linked by this script. Refer to Section 2.6.4 for information on executing this script.

1. Login to UNIX as "root" (the super-user login).

2. Change directories to /usr/BASIC2C by entering the following:

```
cd /usr/BASIC2C    (or the specified NPL RunTime directory)
```

3. Set the BASIC2C_TERM system variable (refer to Section 6.3.1 for valid values). If using the system console, set BASIC2C_TERM as follows:

```
BASIC2C TERM=imon
export BASIC2C TERM
```

NOTE: If the RunTime detects that the BASIC2C_TERM variable is not set and the UNIX TERM variable is set to "ansi", the following prompt is displayed:

"Is this procedure being run on the console on SCO Xenix or UNIX?"

A response of "Y" causes the BASIC2C_TERM value to be set to imon and exported. When the installation procedure has terminated, the BASIC2C_TERM entry is restored to the original value. A "N" response causes the procedure to use the current TERM variable setting.

The security install procedure also automatically sets #TERM=1 during the installation process.

WARNING--SCO UNIX Release 3.2 version 4.0 (later versions are corrected) has a problem with its console drivers such that attempts to display boxes or other extended characters on the console result in alphabetic characters being displayed. SCO has provided a patch for this problem which is available on Niakwa's BBS. Contact Niakwa's Technical Support for more information.

4. Insert the diskette labeled "Master Gold Key" into the diskette drive.

5. At the UNIX prompt, enter:

   ```
   ./xeninst I
   ```

   where I = Install

6. The Gold Key security screen then appears and prompts for the Gold Key diskette. Press Enter to continue or Tab to escape.

NOTE: During the installation, the following warning message appears on the system console once or twice depending on the system being used.

WARNING: error on dev fd (1/1) sector 2396 cmd 0x3 status 0x2000

This message may very from operating system to operating system, but can be safely disregarded.

After a successful installation, the following message appears:

```
"NPL RunTime Gold Key has been successfully installed."
```

NOTE: The ttys file must be established before normal use. Refer to Section 7.4 for more information on this file.

2.6.4 Access Privileges

Once the NPL RunTime and Development Package diskettes have been copied to the hard drive, the RunTime must be configured for proper access privileges. Both the RunTime programs (rtp and rti) and the compiler (b2c) must execute functions that are normally restricted in an UNIX environment. However, UNIX allows programs to execute previously restricted functions if the program has been given super-user privileges. Therefore, Niakwa has provided a shell script program to perform the required configuration steps to give super-user privileges to the compiler (b2c) and the RunTime Programs (rtp and rti) automatically.

setup.nodes

A file named setup.nodes has been included on the Gold Key diskette in the /usr/BASIC2C directory. The setup.nodes shell script file assigns rti, rtp, and b2c super-user access privileges and sets up devices for non-SCO based flavors of UNIX.

NOTE: The setup.nodes shell script must be executed on all installations while the NPL files are located in the /usr/BASIC2C directory.

In addition, Interactive and AT&T UNIX installations should execute this script prior to attempting the installation of the Gold Key security.

To execute the setup.nodes script, follow the steps shown below.

1. Login to UNIX as "root" (the super-user login)
2. Change directories to /usr/BASIC2C by entering the following:
   ```
   cd /usr/BASIC2C
   ```
3. Enter:
   ```
   ./setup.nodes
   ```
4. At the prompt requesting if this is an Interactive or AT&T UNIX install or not, respond appropriately.

NOTE: **This step configures devices for Interactive and AT&T based flavors of UNIX for NPL operation.**

2.6.5 Terminal Configuration

Once the NPL RunTime is fully installed, it is then necessary to configure the system for end-user operation. This typically involves the setup and configuration of the system's terminals for operation with NPL. Detailed information on these configuration issues are covered in Chapter 3.

2.6.6 Recalling the Gold Key Security

The following conditions may require recalling the Gold Key security fingerprint:

- The current hard drive must be reformatted or replaced.
- Before upgrading the operating system.
- Before running the operating system utilities that may affect the operating system kernel or file system (e.g., utilities that rebuild the kernel).
- The RunTime Program must be installed on a different system.
- The RunTime Program must be secured from being used.

NOTE: The NIAKSER.DAT and NIAKSEC.386 files contain important information related to the RunTime security. Avoid moving or deleting these files. For more information on NIAKSER.DAT, refer to Section 2.5.2.

Routine backup and restore procedures should not affect the installed security.

Recall Steps

To recall the Gold Key security on the hard drive, place the original Gold Key diskette in the diskette drive and follow the steps shown below.

1. Login to UNIX as "root" (the super-user login).

2. If the NPL RunTime files have been copied to any directory other than /usr/BA-SIC2C, make sure the NIAKWA_RUNTIME environment variable is set as discussed in Section 2.5.4.

3. Change directories to /usr/BASIC2C (or whatever directory is being used) by entering the following:

```
cd /usr/BASIC2C
```

3. Be sure that the BASIC2C_TERM system variable is set correctly (refer to step 3 in Section 2.6.3).

4. Insert the diskette labeled "Gold Key" into the diskette drive.

5. At the UNIX prompt, enter:

```
./xeninst D
```

   where D = De-install (recall)

   After a successful recall of security, the following message appears:

```
"NPL RunTime Gold Key has been successfully de-installed."
```

2.7 Installing the Upgrade RunTime Package

A special Upgrade RunTime Package is available. This Upgrade RunTime Package is designed so that when it is installed, it is tied to the Gold Key security and serial number of the existing Release III RunTime that is currently installed on the host system's hard drive. This Upgrade RunTime eliminates the need for the old RunTime to be returned to Niakwa. In addition, use of the Upgrade RunTime provides the same serial number and #GOLDKEY value as the current RunTime in use.

The Upgrade RunTime is similar to the Standard RunTime, except for the following:

- The Upgrade RunTime diskettes do not contain the Gold Key security used to pass the RunTime security check.
- The Upgrade RunTime diskette does not contain a NIAKSER.DAT file. This file is updated during the upgrade procedure.

The concept behind the Upgrade RunTime is that the serial number from the currently installed RunTime is extracted by a special upgrade program. This "Serial Number" is then used by the new RunTime to pass the security check.

2.7.1 Requirements for Upgrading

For the upgrade procedure to complete successfully, the following items are required:

- The system being upgraded must have a current Release III (Revision 3.00 or greater) version of the NPL (formerly Basic-2C) RunTime installed (the Gold Key security from this RunTime must be installed on the hard drive).
- The Upgrade RunTime Package number of users must match the currently installed RunTime's number of users.
- Either rtp or rti must be present and executable (rtp is used if both rtp and rti are present).
- All files from the Upgrade RunTime diskettes must be copied to the directory where the existing RunTime files are located (typically /usr/BASIC2C).

- Select the /usr/BASIC2C directory or the NIAKWA_RUNTIME environment variable directory if not set to /usr/BASIC2C.
- The Upgrade RunTime Package must match a valid upgrade option, as shown below, or the upgrade procedure will abort.

The following is a complete list of all NPL RunTime "no-return" upgrade options based upon existing RunTime installation Gold Key revision and type (a Xenix upgrade cannot upgrade a UNIX site and vise versa).

Gold Key Revision	**Old RunTime Type (prior to Release IV)**
3.xx	SCO Xenix
3.xx	ALTOS UNIX
3.xx	SCO UNIX
3.xx	INTERACTIVE UNIX
3.xx	AT&T UNIX

NOTE: The Gold Key revision number RunTime type appears on the original Gold Key diskette label.

2.7.2 How the Upgrade Works

The Upgrade program is used to produce the required NIAKSER.DAT file for the new version of the RunTime being installed. The following steps are performed by the upgrade program:

1. The Upgrade program performs a series of tests to determine if the upgrade should proceed. Various self-explanatory error messages may be generated during this phase. The upgrade procedure is aborted if an error occurs during this phase. Once the error condition is corrected, the Upgrade program can be rerun.

2. All required disk operations are attempted (to ensure access to files, sufficient space on the hard disk, etc.). Again, self-explanatory error messages may be generated during this phase. The upgrade procedure is aborted if an error occurs during this phase. Once the error condition is corrected, the Upgrade program can be rerun.

3. A security check is performed on the upgrade diskette. The procedure is aborted with an error message if the security check fails.

4. Once the above test are done the Upgrade program updates NIAKSER.DAT.

5. The Upgrade Procedure then:

   - Creates a backup copy of NIAKSER.DAT on the upgrade diskette.
   - Deletes rtpold and rtiold if they exist.
   - Renames the existing RunTime programs, rtp and rti to rtpold and rtiold.
   - Renames the Upgrade RunTime programs, RTPUPG and RTIUPG to rtp and rti.
   - Exits with a message of successful completion.

NOTE: All error messages produced by the upgrade program are contained in the script file "upgrade". These error messages may be translated for use by non-English users.

2.7.3 Performing the Upgrade

The following installation instructions assume that drive 0 is the floppy drive and /usr/BASIC2C is the target directory. If the designations are different on the system being used, be sure to use the correct designations instead.

Upgrade Steps

To perform the upgrade, the following steps must be performed:

1. Login to UNIX as "root" (the super-user login).

2. Enter the following UNIX commands to install the NPL Upgrade files from drive 0 to the system’s hard drive:

```
cd /           Select the current directory as root.
umask 0000     Sets protection mode to full access for all users.
```

3. For installing from drive 0 enter:

   for 3-1/2" 720K media

```
tar xvfk /dev/fd0135ds9 720
```

NOTE: For Interactive and AT&T UNIX installation use the following until setup.nodes is run:

```
tar xvfk /dev/dsk/f0q9dt 720
```

   for 5-1/4"" 1.2MB media

```
tar xvfk /dev/fd096ds15 1200
```

   The screen will display a list of files being copied to the system's hard drive by the tar process. This copies the Upgrade RunTime software required to perform the upgrade, to the existing /usr/BASIC2C directory on the hard drive. This must be done before running the upgrade script (upgrade).

NOTE: If the old Release III NPL files are in a directory other than /usr/BASIC2C, the new upgrade files must be moved to that directory prior to going on to step 4.

4. Select the /usr/BASIC2C directory (or the appropriate directory where the NPL files are located) on the hard drive.

```
cd /usr/BASIC2C
```

5 Execute the upgrade program (upgrade):

WARNING--Once the upgrade procedure is complete, the Release IV Upgrade is permanently tied to the original Gold Key diskette and the upgrade diskette contains a copy of the newly-updated version of the NIAKSER.DAT file. The original Gold Key must be saved. Both this diskette and the Upgrade RunTime diskette(s) should be stored together in a safe location as an emergency backup.

Do not write-protect the upgrade diskette(s).

Only the original upgrade diskette(s) may be used to perform the upgrade.

Once the upgrade procedure is successfully completed, it cannot be executed again.

```
upgrade [drive]
```

   where [drive] is the diskette drive in which the Upgrade RunTime is present.

For example:

```
upgrade 0
```

Upon completion of the "upgrade" procedure the following message is displayed:

```
"Upgrade was completed successfully"
```

The Upgrade RunTime Package is now installed. Execution of rtp (or rti if enabled) causes the new RunTime to be executed.

Refer to Section 2.7.2 for details on the operation of the "upgrade" procedure.

NOTE: If an error occurs, please make a note of the error code before calling Niakwa.

The upgrade procedure can only be run once. Once the upgrade has been successfully completed, it no longer executes. However, should the upgrade procedure be terminated due to some error condition, the execute count is not decremented and the upgrade procedure may be rerun once the error condition is corrected.

Once the upgrade procedure is complete, the NIAKSER.DAT file produced during the Upgrade procedure is automatically backed up to the upgrade diskette. The Upgrade RunTime is now permanently tied to the security fingerprint of the original Gold Key (NPL Revision 3.00 or higher).

2.7.4 Recalling the Upgraded RunTime Package

If any of the following conditions arises, it is necessary to recall the upgraded RunTime from the hard drive:

- The current hard drive must be reformatted or replaced.
- Before upgrading the operating system.
- Before running the operating system utilities that may affect the operating system kernel or file system (e.g., utilities that rebuild the kernel).
- The RunTime Program must be installed on a different system.
- The RunTime Program must be secured from being used.

To recall the Upgrade RunTime, the installed Gold Key security of the original Gold Key that is currently installed on the hard drive must be recalled. There are no special steps to be performed with the actual Upgrade Diskettes. Refer to Section 2.6.6 for details on recalling the Gold Key security from the host system's hard drive.

2.7.5 Reinstalling the Upgrade RunTime Package

The following steps should be performed to reinstall the Upgrade RunTime after it has been recalled for any reason.

Reinstallation Steps

1. Copy the files from the original Gold Key diskette as discussed in Section 2.6.2. They are copied to the /usr/BASIC2C directory.

2. If the NPL RunTime files are to be stored in a directory other than /usr/BASIC2C, it is necessary to move them from /usr/BASIC2C to the desired location. For example:

   ```
   mv /usr/BASIC2C/* /usr/NPL
   ```

   Also make sure the NIAKWA_RUNTIME environment variable is set as discussed in Section 2.5.4.

3. Install the Gold Key security from the original Gold Key to the hard drive. Refer to Section 2.6 for details.

4. Mount the latest RunTime Upgrade Diskette and tar all files from the RunTime Upgrade diskettes. If necessary, move them from /usr/BASIC2C to the desired directory as described in step 2 above.

NOTE: This now includes the NIAKSER.DAT file created by the Upgrade procedure.

5. Rename rtp to rtpold.

6. Rename rti to rtiold.

7. Rename RTPUPG to rtp.

8. Rename RTIUPG to rti.

The upgraded RunTime is now reinstalled and operational.

CHAPTER 3

CONFIGURATION

3.1 Overview

NPL applications consist of one or more diskimage files containing compiled NPL programs and data files, at least one start up "BOOT" program, and possibly other associated files. The application software developer must decide where to place these applications within the UNIX file system and how to access them from NPL. The method of setting up NPL applications to operate in a UNIX file system is reviewed in this chapter

Section 3.2 discusses the UNIX file system.

Section 3.3 discusses the location of all NPL software.

Section 3.4 discusses the location of application programs and data.

Section 3.5 discusses the various NPL auxiliary files and their functions.

Section 3.6 discusses configuring additional users for operation within the system.

Section 3.7 discusses configuring the user's workstation.

Section 3.8 discusses the required access privileges.

Section 3.9 discusses invoking the RunTime from a shell script.

Section 3.10 discusses using a menu system to invoke the RunTime from a script file.

Section 3.11 discusses the linking of start-up and menu scripts files.

3.2 The UNIX File System

UNIX uses a hierarchical file system to organize its own system files. Individual directories are used to store the various components of the UNIX operating system.

Each NPL application should be placed in its own directory under the UNIX operating systems. It is recommended that directories for NPL applications be created as subdirectories of the /usr directory. This is because all subdirectories of the /usr directory are considered part of the user file system. Other directories are placed on the root file system, where there is limited space.

NOTE: The BASIC2C directory itself is set up as a subdirectory of /usr.

By establishing a series of subdirectories within the /usr directory, multiple application systems can be organized, allowing for ease of operation, including separate backups of individual systems or directories.

3.3 Location of NPL Software

By default, all NPL software, including the Niakwa Development and RunTime Packages, must be installed in the /usr/BASIC2C directory to insure proper operation of the RunTime and Compiler. This is achieved during the installation of the NPL RunTime and Development Packages.

NOTE: The /usr/BASIC2C directory is the default directory for NPL. If it is necessary to establish a different directory as the default NPL directory, the NIAKWA_RUNTIME environment variable must be set to point to the new default directory. Refer to Section 2.5.4 for more information.

Refer to Chapter 2 for details on installing the Niakwa Development and RunTime Packages.

3.4 Installation of Application Software

The steps described in this section are for installing an example NPL application at an end-user site (as opposed to a development system) under UNIX. After installing the NPL RunTime (refer to Chapter 2 for details), the application software can be installed.

NOTE: When porting an existing system from a Wang 2200 or another supported NPL environment, refer to Chapter 10 for a discussion of porting software to a UNIX system.

If the application is already in UNIX format, perform the following steps:

1. Login to UNIX as "root" (the super-user login)

2. Enter the following UNIX commands:

```
cd /           Select the current directory as root.
umask 0000     Set protection mode to full access for all users.
```

3. Determine and create the directories in which the application program and data files are to be installed using the UNIX mkdir command.

4. Copy the application files into their appropriate directories.

5. Modify the $DEVICE statements in the "BOOT.OBJ" file to reflect the locations of the application program and data files (refer below).

NOTE: When porting an application directly from a Wang 2200, the program files must be compiled to operate under NPL. Refer to Chapter 9 for details on compiling.

UNIX is case sensitive. If the directory is created as /usr/AR, all subsequent access to AR must be in upper case.

Locating Application Programs and Data

As explained in Section 3.2, all application programs and data should be located in separate subdirectories of the /usr directory.

There are several reasons why application software should be segregated from the NPL software, these include:

- For the purposes of backing up only those data files relevant to an individual application system.
- To avoid the frustration of trying to locate one particular file on a directory which is intermixed with several other application files.

For example, when installing an accounts payable system and an accounts receivable system, the directory structure containing these two application systems might appear as:

/usr/AP	Contains the accounts payable main directory with two subdirectories named PROGS and DATA.
/usr/AP/PROGS	Contains the AP subdirectory PROGS with all of the AP application programs.
/usr/AP/DATA	Contains the AP subdirectory DATA with all of the AP application data files.
/usr/AR	Contains the accounts receivable main directory with two subdirectories named PROGS and DATA.
/usr/AR/PROGS	Contains the AR subdirectory PROGS with all of the AP application programs.
/usr/AR/DATA	Contains the AR subdirectory DATA with all of the AP application data files.

3.5 Auxiliary Files

The NPL Development Package contains a series of auxiliary files that the NPL uses to address a variety of functions. This section provides a detailed discussion of these files.

Auxiliary files required for the end user installation may be copied directly from the appropriate NPL Development Package diskette by entering the following UNIX commands:

```
    tar xvfk /dev/fd0135ds9 720 /usr/BASIC2C/filename
```

Alternatively, auxiliary files or modified versions of the auxiliary files can be copied from a development system to a diskette and then to the end-user's system by entering the following UNIX commands:

On the development system:

```
    tar cvf /dev/fd0135ds9 /usr/BASIC2C/filename
```

On the end user system:

```
    tar xvf /dev/fd0135ds9 /usr/BASIC2C/filename
```

3.5.1 The "ENABLED" File

For the Interpretive RunTime (rti) to operate, it is necessary to install a special file, called "ENABLED". This file is located on the NPL Compiler diskette and must be installed in the /usr/BASIC2C or appropriate NPL files directory of the host system's hard drive.

If the Interpretive RunTime is invoked without the presence of the ENABLED file in the current, NIAKWA_RUNTIME set, or /usr/BASIC2C directory, the message:

Interpreter not enabled

appears, and RunTime execution is canceled.

Installing ENABLED in the end user's /usr/BASIC2C or appropriate NPL files directory enables the Interpreter for all applications.

HINT: It is also possible to install the ENABLED file only in the current directory of specific applications. This allows the Interpreter to be used with some applications while preventing its use with other applications.

NOTE: Use of the Interpreter allows the end-user access to several functions which, if used improperly, could be damaging. These are the RESET and STEP functions of the HELP processor and the HALT key (refer to Chapter 11 of the NPL Programmer Guide for more information on the HELP processor). These functions can be suppressed under program control by use of the $OPTIONS system variable. For a detailed discussion of the $OPTIONS system variable, refer to the NPL Statements Guide, $OPTIONS.

Enabling the Interpretive RunTime program at the end user's site requires that the end user execute the Niakwa End User Support Only License Agreement.

3.5.2 Keyboard Files

The keyboards of many terminals used by NPL are not completely compatible with the NPL character set. These keyboard differences are resolved by the use of a look up table which translates keys received from the keyboard to hex codes expected by NPL programs. The standard built-in defaults for keyboard remapping are present within the RunTime and should prove adequate for most applications.

NOTE: The method the NPL RunTime program uses to handle keyboard character translation under an UNIX operating system is different than under other versions of NPL. The presence of a keyboard translation table file is required or the NPL RunTime may hang the executing terminal.

Should modifications be required, they can be made by using the NPL Keyboard Translation Tables Editor. This utility will modify the various disk files named KEYBOARD.xxxx, where xxxx is the proper suffix of the terminal in use. NPL determines the proper value of this suffix from the value of the UNIX system variable TERM, or from the value of the BASIC2C_TERM system variable. NPL first searches for the KEYBOARD.xxxx file in the currently selected directory when executed. If the file is not found, the RunTime then searches the NIAKWA_RUNTIME set directory and finally the /usr/BASIC2C directory.

NOTE: If the file is not found, the keyboard cannot be properly configured and will not function properly.

The NPL Development Package provides a series of KEYBOARD.xxxx files on the NPL Terminal Support diskette. These files are used by NPL for use with the various terminals supported by the RunTime. Be sure to use the properly named KEYBOARD file for the terminal in use if modifications are made to the keyboard translation table. Refer to Chapter 6 for the appropriate file names.

The above files are also used by the NPL RunTime with the NPL REDIRECT feature of the HELP facility, thus providing for a variety of terminals that may be used for remote communication and support with the end user's site.

To modify any of the default keyboard translation tables:

1. Create the modified KEYBOARD.xxxx file(s) on the development system.
2. Copy the file(s) to a diskette (refer to Section 3.5 for details).
3. Copy the file(s) to the end user's system as part of the standard application installation procedure (refer to Section 3.5 for details).

Refer to Chapter 13, of the NPL Programmer's Guide, for details on the use of the NPL Keyboard Translation Tables Editor utility. Refer to Appendix D of the NPL Programmer's Guide for details on supported terminals. Also, refer to Chapter 6 of this Supplement for details of the IBM-compatible PC keyboard characteristics, for the console terminal under the UNIX implementation of NPL.

WARNING--If reinstallation of the Terminals Support Files Diskette is necessary (for new versions), any changes made to the above files are overwritten. Consequently, a backup copy of all customized files should be made at the time of their creation and stored in a safe place.

3.5.3 Screen and Font Files

The NPL character set is not fully supported on all the terminals supported by NPL. These screen differences are resolved by using a look up table which translates the UNIX screen characters to screen characters expected from a NPL program.

NOTE: The method that the NPL RunTime program uses to handle screen character translation under an UNIX operating system is different than under other versions of the NPL. The presence of a screen translation table file is required.

Should modifications be required, they can be made by use of the NPL Screen Translation Table Editor. This utility allows modification of the various disk files named SCREEN.xxxx, where xxxx is the proper suffix of the terminal in use. NPL determines the proper value of this suffix from the value of the UNIX system variable TERM, or from the value of the BASIC2C_TERM system variable. NPL first searches for the SCREEN.xxxx file in the currently selected directory when executed. If the file is not found, the RunTime then searches the NIAKWA_RUNTIME set directory and finally the /usr/BASIC2C directory.

NOTE: If the file is not found, the screen translation cannot operate correctly and may hang the terminal.

In addition, many NPL terminals support the use of downloadable fonts. Downloading of the screen character set to the Wyse 60, Wyse 150, Wyse 160, Wyse 370, Altos V, and VT220 series terminals is performed through the files WYFONT.WY6, W370FONT.80, W370FONT.132, VTFONT.AL5 and VTFONT.VT2 respectively. Should modifications be required, the NPL Font Table Editor utility program allows the programmer to create or modify these files. Refer to Chapter 6 of this supplement, Chapter 13 and Appendix D of the NPL Programmer's Guide for more information.

The Screen and Font files are also used by the Niakwa RunTime with the NPL REDIRECT feature, thus providing for a variety of terminals that may be used for remote communication to the end user's site.

To modify any of the default font or screen translation tables:

1. Create the modified FONT or SCREEN.xxxx file(s) on the development system.

2. Copy the file(s) to a diskette.

3. Copy the file(s) to the end user's system as part of the standard application installation procedure.

   Refer to the NPL Statements Guide, $SCREEN, for details on $SCREEN. Refer to Chapter 13 of the Programmer's Guide for details on the NPL Screen Translation Tables Editor. Refer to Chapter 6 for more details on UNIX specific screen characteristics under the NPL.

WARNING--If reinstallation of the Terminals Support Files Diskette is necessary (for new versions), any changes made to the above files are overwritten. Consequently, a backup copy of all customized files should be made at the time of their creation and stored in a safe place.

3.5.4 Printer Control Values

Since printer control protocol standards are virtually non-existent, printer control specifications vary dramatically from printer to printer. The print control feature of the NPL has built-in default control codes (refer to Section 4.6 for actual default values) which should work well with the IBM-compatible PC Graphics Printer (Epson MX-80, or equivalent).

If other types of printers are in use on the system, use the NPL Edit Printer Control Codes utility to define a PRINTCTL.TBL file of printer control values so that the functions listed on the Printer Control screen operate correctly.

When the RunTime program is executed, it looks for the PRINTCTL.TBL file in the current directory. If the file is not found, the RunTime then searches the NIAKWA_RUNTIME set directory and finally the /usr/BASIC2C directory. If the file is not found, the built-in defaults of the RunTime are used.

Refer to Chapter 13 of the NPL Programmer's Guide for details on the operation of the NPL Edit Printer Control Codes utility.

3.5.5 Error Descriptions

The RunTime diskette contains four files, labeled ERRORMSG.HLP, ERRORMSG.IDX, RTIXERR.HLP and RTIXERR.IDX, which are copied to the /usr/BASIC2C directory on the hard disk when the RunTime is installed. The ERRORMSG files are used to display the NPL error code messages when an error occurs. The RTIXERR files are used to display the native operating system error code message when an error occurs. The files with the extension of .HLP contains the literal description of each NPL or native operating system error code, while the .IDX file is the indexed help files required to "find" the proper entry in the descriptive files.

The ERRORMSG.HLP and RTIXERR.HLP files can be modified by the developer so that different error descriptions can be displayed. This is particularly useful for distributors of non-English applications. However, if the ERRORMSG.HLP and RTIXERR.HLP files are modified, they must be processed by the NPL Indexed Help File Processor utility (refer to Section 13.16 of the NPL Programmer's Guide for details). Refer to Chapter 11 of the NPL Programmer's Guide for details on indexed help files.

WARNING--If reinstallation of the RunTime Package is necessary (for new versions or Upgrades), any changes made to the above files are overwritten since these files are contained on the NPL RunTime Package diskettes. Consequently, a backup copy of all customized files should be made at the time of their creation and stored in a safe place.

3.5.6 Terminal Identification

Terminal identification is determined by the NPL RunTime by interrogating the ttys file in the Niakwa files directory or in the /etc directory.

If NPL cannot find the ttys file, the error message "Cannot Determine Terminal Number" is displayed and the RunTime cannot execute. If the ttys file does not exist, it must be created.

To help in the creation of the ttys file, the RunTime Development Package, Terminal Support Files Diskette, contains an example file called ttys.exp. The ttys.exp file can be copied to the ttys file and then modified to fit the needs of the installed system.

For further details on the ttys file and terminal type identification refer to Section 7.4.

3.5.7 Additional End-User Security (#GOLDKEY)

The GOLDKEY.OBJ program allows developers to determine the #GOLDKEY number for any NPL RunTime based on the Gold Key serial number without having to physically open the RunTime Package.

This program can be found on the NPL Development Package Compiler diskette and, once installed, can be run as any other NPL program.

When executed, the GOLDKEY program prompts for the Gold Key serial number as shown below.

```
    Enter Serial Number (1 - 65535) to convert to #GOLDKEY:
```

Once the serial number is entered, the program returns the correct #GOLDKEY code number that is necessary for some application security programs.

3.6 Configuring Additional Users for Operation

This section discusses the procedure of establishing users in an UNIX environment.

3.6.1 Adding a User to UNIX

The commands used for adding users to a UNIX system vary between different flavors of UNIX. Refer to the appropriate UNIX manuals for information on how to add users.

3.6.2 The User Directory

The procedure for adding a user to an UNIX system includes the creation of a new subdirectory in the user filesystem. The creation of this directory is typically created automatically when the user is added. This directory is the user’s "HOME" directory. This means that whenever a user logs in, the current directory is always set to the users "HOME" directory.

3.6.3 The .profile File

The procedure for adding a user to an UNIX system includes the creation of a .profile file for each user. This .profile file is normally placed in the user’s HOME directory. This .profile file must contain certain statements required for proper execution of the NPL RunTime program. Refer to Section 2.3.2 for details on the configuration of the .profile file.

3.7 Configuring User's Workstations

UNIX is a multi-user NPL environment and as such NPL supports many terminals for use with the NPL RunTime. Refer to Chapter 6 for details on configuring terminals for use with NPL under UNIX.

3.8 Required Access Privileges

UNIX access privileges must also be set to allow end user access to developer created application programs and files as necessary. Rights are typically assigned by the super-user through the UNIX "chmod" command. Refer to the UNIX documentation for further information on the use of the chmod command and user access privileges.

NOTE: Access privileges for the NPL RunTime files as discussed in Section 2.6.4.

3.9 Executing the Runtime from a Shell Script

Under the UNIX implementation of NPL, it is advantageous to invoke the RunTime from a "procedure file". A procedure file is an ASCII text file containing a list of commands as they would be entered by the user. This file would typically select the proper directory and then invoke the RunTime with the proper "BOOT" program name. This simplifies operations for the end user. A procedure file in UNIX is called a "shell script". A UNIX text editor (e.g., vi, ed, etc.) can be used to create a "shell script" file.

HINT: This is also a very important technique to use, since more than one user would have to remember the directory designations and "BOOT" program name to use.

For example, assume that a NPL application has been set up in the directory /usr/AR and that the name of the "BOOT" program is "ARBOOT". A typical shell script file for executing this application would be:

```
(cd /usr/AR; rtp ARBOOT)- using the Non-Interpretive RunTime
```

or

```
(cd /usr/AR; rti ARBOOT)- using the Interpretive RunTime
```

NOTE: The semicolon is used to delimit multiple commands on a single command line.

rtp or rti is in lowercase. An attempt to execute RTP or RTI results in "program not found" (UNIX is case sensitive).

To be recognized as a shell script, the file must be given execute access privileges from the chmod command. For example, to make a shell script called "arstart" executable, enter the following UNIX command:

```
chmod a+x arstart
```

Upon exiting the AR application, the user's current directory is unchanged. This is because the "cd" statement in the above example is only in effect for the duration of the shell script.

An alternative method of setting up the shell script would be:

```
cd /usr/AR
rtp ARBOOT     (or rti ARBOOT)
```

However, this method would leave the user's current directory as /usr/AR after exiting rtp (or rti), until the end of the shell script. This may be undesirable if commands follow the rtp (or rti) progname command.

A sample shell script is provided for executing the NPL utilities as part of the NPL Development Package. This file is called "util". To invoke this shell script enter "util" from any sub-directory.

3.10 Executing the RunTime from a Menu System

Under UNIX, it is very simple to create shell script files which can be used to invoke the NPL RunTime. This method means that the operator does not have to remember which directory to select and what "BOOT" program name to use. However, if there are many different applications, the operator may still have difficulty remembering the names of the various shell script files. This "problem" can easily be resolved by use of a menu instruction file.

3.10.1 Creating the Menu Instruction File

The menu instruction file contains a list of descriptions of the various applications available along with the associated shell script name. This file can be created by using one of the standard UNIX text editors such as vi. This menu file may be displayed on the operator screen by means of the UNIX "cat" command. The operator then simply enters the name of the shell script for the application desired.

For example, assume that a given installation has two applications, AR and AP. Shell script files for both have been created (much like in the example in Section 3.9). The names of these shell script files are arstart and apstart. A typical menu instruction file may contain:

```
Enter the name of the program to run:
Accounts Receivables              arstart
Accounts Payable                  apstart
```

This menu file may be given any convenient name. For purposes of this example, assume the name used is "APPMENU".

3.10.2 Displaying the Menu Instruction File

The menu instruction file should be displayed in two places. The first is immediately following the user login. The second is following the completion of any application (redisplay the menu so other options can be selected).

To display the menu instruction file when a user initially logs in, the following line is added to the user's .profile file (refer to Section 2.3.2 for details on .profile).

```
cat APPMENU
```

where APPMENU is the name of the menu instruction file.

To display the menu instruction file following completion of an application, add a line to the end of the start up shell script file for the application:

```
(cd /usr/AR; rtp ARBOOT)
cat APPMENU
```

where APPMENU is the name of the menu instruction file.

For this to work properly, the shell script statement which invokes rtp (or rti) must be coded as a subprocess. This ensures that the "current" directory is properly set when the cat APPMENU instruction is executed.

3.10.3 An Example Menu Instruction File

An example menu instruction file is provided with the NPL Development Package. This file is named "utilinst" and can be found in the /usr/BASIC2C directory following installation of the NPL Development Package.

3.11 Linking Start-Up and Menu Script Files

At this point additional users have been created on the system, the terminals have been configured, start up shell scripts have been created, and a menu system has been established. Now it is necessary to give the user access to the NPL applications. Although this can be done by many methods under UNIX, the best method is to "link" the appropriate start up shell scripts and menu instruction files into the user's HOME directory. This can be done by using the UNIX "ln" command. The ln command creates a directory entry in the user's HOME directory for files specified.

NOTE: This does not make a copy of the file. Rather, only a second pointer to the same physical file is established. This means that if any changes are made to a "linked" file, the changes are immediately available to all "links" to the file.

In the following example, assume that the start up shell scripts are called arstart in directory /usr/AR and apstart in directory /usr/AP. Assume that the menu shell script is called APPINST and is located in the /usr/BASIC2C directory.

To link the start up shell scripts and menu instruction files, enter the following from the UNIX shell:

```
ln /usr/AR/arstart /usr/username
ln /usr/AP/apstart /usr/username
ln /usr/BASIC2C/APPINST /usr/username
```

where username is the name of the user being set up.

Login as super-user to perform the above tasks. Also, be sure to set the access mode on the "linked" files so that the user can access them. For example:

```
chmod a+x /usr/username/arstart
chmod a+x /usr/username/apstart
chmod a+r /usr/username/APPINST
```

The shell script files used to invoke NPL are given execute access while the menu instruction file is given only read access. The menu instruction file is never "executed", rather it is "read" by the cat command.

If different users have different sets of applications that they may execute, it may be advisable not to "link" the menu instruction file, but to copy it instead, and modify it to the particular needs of the user. However, it is advisable to "link" the shell scripts, thus ensuring that all users access the same application through the same procedure (and that changes to the shell scripts affect all users immediately).

CHAPTER 4

RUNTIME OPERATION

4.1 Overview

This chapter covers the general operation of the NPL RunTime under Intel UNIX operating systems. The chapter discusses various methods of starting and exiting the RunTime program, the available RunTime startup options, memory considerations, and the default printer control values of NPL under UNIX.

Section 4.2 discusses the two RunTime programs available.

Section 4.3 discusses the general startup form of the RunTime.

Section 4.4 discusses the use of various startup options available in the RunTime.

Section 4.5 discusses the determination of available memory by the RunTime.

Section 4.6 discusses the default printer control values of the RunTime.

Section 4.7 discusses the various methods of exiting the RunTime under UNIX.

4.2 rtp Versus rti

Included in the RunTime package are two RunTime programs: One program is Non-Interpretive, (rtp), and the other Interpretive (rti).

The Non-Interpretive version allows for execution of NPL object code, without any capabilities for "immediate mode" command entry or functions.

The Interpretive RunTime program allows for full development capabilities, such as program text editing and debugging. The interpretive version does require more memory. Refer to Section 2.2 for details on memory requirements.

NPL developers may choose whether or not end-user sites have interpretive capabilities. Installation of the "ENABLED" file is necessary to allow the Interpretive RunTime Program to execute. Refer to Section 3.5.1 for details on the "ENABLED" file.

NOTE: Although interpretive capabilities at the end-user site may be useful for support purposes, these capabilities allow an end-user access to the developer's source code, which may not be desirable.

Use of the Interpreter at end-user sites requires execution of the End-User Support Only License Agreement. Refer to Section 3.5.1 for details on the "ENABLED" file required for operation of the Interpretive RunTime program.

4.3 Starting the NPL RunTime

Several methods are available to begin the execution of either RunTime. This section discusses the RunTime's general startup option form.

4.3.1 Starting From the Command Processor

When in the UNIX command processor (typically a korn shell), the general form of a command line to execute either RunTime program is:

```
     rtp [option]...[progname]     (non-interpretive version)

     rti [option]...[progname]     (interpretive   version  )
```

where:

```
     option    = RunTime startup option or combination of
                 startup options. Options can be entered in
                 either upper or lower case. For example -d=32
                 or -D=32 (this applies only to the single let-
                 ter after the "-"--not any filenames that may
                 follow which are case-dependent).

     progname =  the filename of the NPL object program to be
                 executed.  If no progname is specified, the
                 RunTime Program assumes the program
                 "BOOT.OBJ" is to be executed. If an extension
                 is not supplied, NPL assumes an extension of
                 ".OBJ" on the progname (the filename is oper-
                 ating system-dependent).
```

4.3.2 Starting from Shell Script Files

Shell script files can be written to directly execute the RunTime. The script file can perform the various commands necessary to invoke the RunTime for a specific application.

Refer to Section 3.9 for additional information on starting the RunTime from a script file.

4.3.3 Starting from a Menu

For a more user-friendly approach to starting an NPL application, the RunTime program can be started from a menu system. Many third-party products can be used for this purpose. Refer to Section 3.10 for additional information on starting the RunTime from a menu system.

4.4 Command Line (Start-up) Options

The NPL RunTime environment is set up internally by the rtp and rti programs upon execution. The following section discusses the available RunTime startup options which may be specified upon execution of the RunTime. These options allow for modification of the default RunTime environment.

4.4.1 -B (Background Partition)

The -B option indicates that the NPL RunTime is to begin operation in a background partition.

```
    rtp [-B=xx]

    rti [-B=xx]
```

where:

```
     xx =  the #PART value to be assigned.
```

For example:

```
rti -B=40
```

Starts the RunTime in a background partition with a #PART value of 40.

Refer to Section 8.3 for details on background partitions.

Refer to Section 2.4 of the NPL Programmer’s Guide for a complete discussion of the -B option.

4.4.2 -D (DET Entries)

The -D RunTime option allows a programmer to specify the number of device equivalence table entries. The number of DET entries may be a range of 16 to 255. If -D is not specified, the default of 16 DET entries is used.

```
     rtp [-D=nnn]

     rti [-D=nnn]
```

where:

```
      nnn =  the number of device equivalence table entries.
```

For example:

```
rti -D=32
```

specifies that 32 DET entries are available.

Special Considerations for -D Option

When using the -D option, be aware of the following operational considerations.

- Each DET entry above 16 results in the use of additional memory. The amount of memory each additional DET entry requires is 64 bytes. This memory is deducted from the available user partition.

NOTE: This memory requirement may increase with future revisions of NPL.

- Open file limits are set during the installation of the UNIX operating system. Refer to the UNIX system documentation for details on installing and reconfiguring the UNIX environment.

Refer to Section 2.4 of the NPL Programmer’s Guide, for a complete discussion of the -D option.

4.4.3 -G (Graphics Mode)

The -G option is not supported for use under UNIX.

4.4.4 -H (Handle Table Size)

The handle table is an internal table used by the RunTime to translate two-byte p-code pointers into four-byte memory addresses. Handle table entries are created at program resolution time and each handle table entry requires 4 bytes of memory. An entry is required for:

Every unique variable
Each unique line number
Each DO/ENDDO group
Each internal DEFFN'
Each loop construct
Each PROCEDURE or FUNCTION

If -H is not specified, the RunTime program allocates a small amount of memory for the handle table and expands it as required.

```
     rtp [-Hx]

     rti [-Hx]
```

where:

```
     x = the amount of space to be allocated for the
         handle table, in units of 4K bytes

         or

         the number of handle table entries, in units
         of 1024 bytes.
```

For example:

```
rtp -H15
```

allocates a handle table of 60K bytes (4K x15) or 15360 entries (60 x 1024/4)..

HINT: A value of one (1) or two (2) for the handle table is typically sufficient.

Refer to Section 2.4 of the NPL Programmer's Guide for a complete discussion of the -H option.

4.4.5 -K (Mouse Support)

The -K option is not supported under UNIX.

4.4.6 -L (Leave Overhead Memory)

The -L option is not supported under UNIX.

4.4.7 -M (XMS Memory)

The -M option is not supported under UNIX.

4.4.8 -P (Pre-Boot)

The -P RunTime option allows for a "pre-boot" configuration program. If the -P option is specified on the RunTime command line, NPL loads and executes a "pre-boot" program.

```
     rtp[-P[filename]]

     rti[-P[filename]]
```

where:

```
     filename = the filename of the pre-boot program to be
                loaded.
```

For example:

```
rti -PSTARTUP
```

specifies that the pre-boot program STARTUP.OBJ is to be executed.

Refer to Section 2.4 of the NPL Programmer's Guide for a complete discussion of the -P option.

4.4.9 -R (Remote Control)

The -R option is used to force the RunTime to use terminal control codes instead of direct video mapping. This option serves no real purpose under UNIX since all ports on the system automatically assume they must use terminal control codes.

Refer to Section 2.4 and 2.9 of the NPL Programmer's Guide for a complete discussion of the -R option.

4.4.10 -S (Separate Program Segments)

The -S option performs no operation under UNIX.

Refer to Section 2.4 of the NPL Programmer's Guide for a complete discussion of the -S option.

4.4.11 -T (Terminal/Port Number)

The -T option is used for specifying a specific #TERM value.

```
     rtp [-T=xx]

     rti [-T=xx]
```

where:

```
     xx = a numeric-constant which indicates which terminal
          number to use (overrides default terminal number
          determination).
```

For example:

```
rtp -T=2
```

Starts the RunTime overriding the default #TERM value with a #TERM value of 2.

HINT: This option is particularly useful if the terminal port in use has no corresponding entry in the /usr/BASIC2C/ttys file and the "Cannot Determine Terminal Number" message appears.

NOTE: Assignment of #TERM values using the /usr/BASIC2C/ttys file (the default method) assures unique #TERM values are defined for each terminal. The -T option should be used with caution since the RunTime does not detect duplicate #TERM values when the -T option is used.

Refer to the Section 2.4 of the NPL Programmer's Guide for a complete discussion of the -T option.

4.4.12 -U (UMB Memory)

The -U option is not supported under UNIX.

4.4.13 The -X (Load External Library)

The -X option is not supported under UNIX. For details on implementing external subroutines under UNIX, refer to Chapter 11.

4.4.14 The BOOT Program

The RunTime program assumes that the first program to be executed is not in an NPL diskimage, but rather is a native operating system file. This first program can be thought of as a "boot" program. The boot program generally sets up or customizes the Device Equivalence Table and runs a start-up program in a diskimage.

The RunTime looks to the command line for the name of the BOOT file to load. If no boot file is specified, the RunTime looks in the current directory for a file called BOOT.OBJ. If this file is not found, the RunTime starts, but displays an error message indicating that a BOOT file was not found.

Refer to section 4.3 for an example of the general form of starting the RunTime with a BOOT file.

The last statement in the boot program typically is a LOAD RUN < progname > , which loads the first program to execute in a selected diskimage.

Refer to Section 2.4 of the NPL Programmer's Guide for more information on the BOOT file.

4.5 Available Memory

Under UNIX, the amount of memory available to the application is not limited by the amount of memory physically present on the host system (provided certain minimum memory requirements are met). However, frequently exceeding the physical memory available significantly degrades performance.

Refer to Section 8.4 for additional details of memory management by the RunTime under UNIX.

4.6 Default Printer Control

Printer control is a function of the RunTime's HELP processor and may be accessed whenever the NPL HELP processor is active. Refer to Chapter 11 of the NPL Programmer's Guide for details on the HELP processor. This section discusses the default RunTime printer control values. For more information on the use of printer control, refer to Section 3.5.

4.6.1 Default Values

The tables below list the default values built into the RunTime program for UNIX. These control codes should work sufficiently for most printers which support default Epson dot matrix printer codes. The following codes may need to be changed depending on the type of printer.

The codes in this table are hex codes, except for single ASCII characters preceded by an equal, "= ", sign.

For example:

```
    1B=A181B=2 is the equivalent of 1B41181B32
```

This technique is supported for on-line entry of control codes during execution of print control. This avoids the need to look up the hex codes for ASCII control sequences.

```
Characters per inch option
    10             1412
    N/A
    N/A
    16             140F

Lines per inch options
    3              1B=A181B=2
    4              1B=A121B=2
    6              1B=AOC1B=2
    8T             1B=A091B=2

Line Feed
    OA

Form Feed
    OC
```

4.7 Exiting from the RunTime Program

There are several methods available to either the programmer or the end-user for exiting the RunTime (and, consequently, the application it is executing). In most cases, exiting the RunTime returns the system to the point at which the RunTime was invoked. That is, if the RunTime was started from a UNIX command processor, exiting returns the system there. If the RunTime was started from a menu, exiting returns to the same menu. A discussion of the various exiting methods follows.

Refer to Section 6.8 for details on command default keyboard equivalences.

4.7.1 Exiting Under Program Control

The RunTime may be exited under program control by any one of the following events:

- Execution of the NPL "END" or "STOP" statement. With either of these statements, a ":" is placed on the screen with the cursor immediately following. When in the Non-Interpretive RunTime, pressing CANCEL or HELP and the KILL RUNTIME option at that point terminates the RunTime. Refer to Chapter 11 of the NPL Programmer's Guide for a complete discussion of the NPL HELP processor.

- By program logic which falls through the logical end of the program. In this event, the RunTime automatically exits without further action required. This is only true for the Non-Interpretive RunTime.
- If an application error condition is encountered, the Non-Interpretive RunTime is exited.
- By execution of a $END statement. This statement causes the RunTime program to end operation, returning control to UNIX.

HINT: This is the preferred method of exiting under program control.

4.7.2 Exiting using the HELP Key

The end-user may call for cancellation of program execution using the NPL HELP key. Depression of the HELP key causes current program execution to be suspended, the screen is saved and the HELP screen is displayed, with various options. The end-user may, at this point, select the KILL RUNTIME option to terminate program execution. If the LEAVE HELP option is selected, the screen is restored and program execution resumes.

The HELP key is active only when the application program is polling for keyboard input (i.e., executing a KEYIN, INPUT or LINPUT). The HELP key with the KILL RUNTIME option can be thought of as a limited 2200 RESET key. For full details on the HELP key, refer to Chapter 11 of the NPL Programmer's Guide.

4.7.3 Exiting using the Interrupt Key

UNIX provides its own methods for terminating program execution. The interrupt key (Ctrl-C) termination method is one of these. Upon depression of the interrupt key, the usual NPL HELP display is provided and program execution may be terminated or resumed in the same methods as described for the HELP key.

NOTE: The interrupt key only takes effect when the application program issues a disk I/O request (i.e., DATALOAD, DATASAVE, DATALOAD DC, etc.). If the interrupt key is pressed when the application is waiting for keyboard input, it is treated as a normal keystroke.

Refer to Chapter 6 for a description of the interrupt key sequence for specific Niakwa supported terminals.

CHAPTER 5

DEVICE SUPPORT

5.1 Overview

This chapter discusses devices supported under the UNIX implementation of NPL. Included in this discussion are any special requirements or implications for the programming of NPL code under the UNIX operating environment.

Section 5.2 discusses supported diskette devices.

Section 5.3 discusses diskimage files.

Section 5.4 discusses supported monitors/controllers.

Section 5.5 discusses printer devices.

Section 5.6 discusses mouse support.

Section 5.7 discusses the use of serial devices.

Section 5.8 discusses the support of a tape drive.

Section 5.9 discusses math co-processor support.

Section 5.10 discusses the default device equivalences.

NOTE: The screen and keyboard characteristics of all supported terminals not specific to the operating system are discussed in Appendix D of the NPL Programmer's Guide. All terminals that are specific to the UNIX operating system are discussed in Chapter 6 of this Supplement.

5.2 Diskette Devices

NPL under UNIX has the ability of reading and writing "raw" diskettes in a variety of formats. The table table in Section 5.2.1 lists all supported formats.

NOTE: $FORMAT DISK is not supported in any format. Attempts to execute $FORMAT DISK result in an I93 error. As a result, diskettes must be preformatted before use with NPL under UNIX.

The 320K raw format is not supported under NPL for UNIX.

The 360K is a 5-1/4" raw format compatible with other NPL versions which support this media type and is partially compatible with the Wang 2200/CS. Refer to Section 5.2.2 for details.

The 720K raw format for 5-1/4" diskettes is compatible with other NPL versions which support this media type. Refer to Section 5.2.2 for details.

The 1.2MB is a 5-1/4" raw format compatible with other NPL versions which support this media type. Refer to Section 5.2.2 for details.

The 720K raw format for 3-1/2" diskettes is compatible with other NPL versions which support this media type. Refer to Section 5.2.2 for details.

The 1.44MB and 2.88MB are 3-1/2" raw format diskettes and are compatible with other NPL versions which support these media types. Refer to Section 5.2.2 for details.

NOTE: Refer to Appendix D for more information on "raw" device compatibility between current NPL-supported operating environments.

5.2.1 Naming Conventions

The following table displays the naming conventions and revisions of the RunTime required for the diskette devices supported under the Release IV UNIX implementation of NPL.

Media Size	Type Format	Naming Convention
5-1/4" media	360K	/dev/fd048ds9
	720K	/dev/fd096ds9
	1.2MB	/dev/fd096ds15
3-1/2" media	720K	/dev/fd0135ds9
	1.44MB	/dev/fd0135ds18
	2.88MB	/dev/fd0135ds36

Under UNIX, each diskette drive is accessed by special device names which indicates both which drive to access and what type of media to expect in the drive (the device name may also be dependent on the flavor of UNIX being used). The drives are designated drive "0" and drive "1". For example, the device name /dev/fd048ds9 refers to drive "0" and /dev/fd148ds9 refers to drive "1".

NOTE: The designation of drive "0" is used in this manual for purposes of example.

UNIX Diskette Device Names

UNIX uses two types of special file name to access each type of diskette. The NPL RunTime Program requires that both types of special file name be properly established. Most of these special file names are established by the UNIX installation procedure.

The remainder of this section provides background information on these special file names.

The first type of special file is the "block" type special file. This is used primarily by standard UNIX I/O routines.

The second type of special file is the "character" type. This is used for access on a character by character basis by certain UNIX routines. In the UNIX documentation, this character type of special file is sometimes referred to as a "raw" file. Access to devices using this type of special file is sometimes referred to as "raw" access. The term "raw" in this context is not to be confused with the term "raw" diskette as used in reference to NPL diskettes. In UNIX documentation, the term "raw" means access to the character type device. In the NPL documentation, a "raw" diskette is one which contains an NPL catalog and index as its primary file structure, rather than the native operating system file structure.

For each type of diskette, both a block and character special file name are set up.

The device names for the character special files are the same as for the corresponding block special files, except for the addition of an "r" (for "raw") at the start of the file name. For example, the special file names for a 5-1/4", 1.2MB diskette on drive 0 are:

```
/dev/fd096ds15   Block type file for a 1.2MB diskette.
/dev/rfd096ds15  Character type file for a 1.2MB diskette.
```

NOTE: These files may be set up with read only access for users other than the super-user. If users other than the super-user are going to require the ability to write to raw diskettes, the access rights to these files should be modified with the UNIX "chmod" command. For example, for use with the 1.2MB drive:

1. Login as "root"

2. Enter:

```
chmod a+w /dev/rfd096ds15 /dev/fd096ds15
```

5.2.2 Supported Media

The following section discusses the various NPL supported media.

NOTE: The default device equivalence for NPL address D10 is /dev/fd096ds15.

5-1/4" Media

320K Media

320K raw diskettes are not supported under the Release IV version of NPL.

360K Media

The $DEVICE clause "/dev/fd048ds9" defines a raw diskette as 360K format. This format is defined as a 360K, 5-1/4" double-sided, double density diskette containing 40 tracks per side, 9 sectors per track, and 512 bytes per sector.

For example:

```
0010 $DEVICE(/D10)="/dev/fd048ds9" : REM 360K raw media on drive 0
0010 $DEVICE(/D10)="/dev/fd148ds9" : REM 360K raw media on drive 1
```

720K Media

The $DEVICE clause "/dev/fd096ds9" defines a raw diskette as 720K format. This format is defined as a 720K, 5-1/4" double-sided, quad density diskette containing 80 tracks per side, 9 sectors per track and 512 bytes per sector.

For example:

```
0010 $DEVICE(/D10)="/dev/fd096ds9" :REM 720K raw media on drive 0
0010 $DEVICE(/D10)="/dev/fd196ds9" :REM 720K raw media on drive 1
```

NOTE: The 5-1/4" 720K diskette format is not supported by all UNIX platforms.

1.2MB Media

The $DEVICE clause "/dev/fd096ds15" defines a raw diskette as 1.2MB (1200K) format. This format is defined as a 1.2MB, 5-14" double-sided, high density diskette containing 80 tracks per side, 15 sectors per track and 512 bytes per sector.

For example:

```
0010 $DEVICE(/D10)="/dev/fd096ds15" :REM 1.2MB raw media on drive 0
0010 $DEVICE(/D10)="/dev/fd196ds15" :REM 1.2MB raw media on drive 1
```

3-1/2" Media

720K Media

The $DEVICE clause "/dev/fd0135ds9" defines a raw diskette as 720K format. This format is defined as a 720K, 3-1/2", double-sided, double density diskette, containing 80 tracks per side, 9 sectors per track and 512 bytes per sector.

For example:

```
0010 $DEVICE(/D10)="/dev/fd0135ds9" :REM 720K raw media on drive 0
0010 $DEVICE(/D10)="/dev/fd1135ds9" :REM 720K raw media on drive 1
```

1.44MB Media

The $DEVICE clause "/dev/fd0135ds18" defines a raw diskette as 1.44MB (1440K) format. This format is defined as a 1.44MB, 3-1/2", double-sided, high density diskette, containing 80 tracks per side, 18 sectors per track and 512 bytes per sector.

For example:

```
0010 $DEVICE(/D10)="/dev/fd0135ds18" :REM 1.44MB raw media on drive 0
0010 $DEVICE(/D10)="/dev/fd1135ds18" :REM 1.44MB raw media on drive 1
```

2.88MB Media

The $DEVICE clause "/dev/fd0135ds36" defines a raw diskette as 2.88MB (2880K) format. This format is defined as a 2.88MB, 3-1/2", double-sided, high density diskette, containing 80 tracks per side, 36 sectors per track and 512 bytes per sector.

For example:

```
0010 $DEVICE(/D10)="/dev/fd0135ds36" :REM 2.88MB raw media on drive 0
0010 $DEVICE(/D10)="/dev/fd1135ds36" :REM 2.88MB raw media on drive 1
```

5.2.3 Determination of Media Type

The media mounted in the diskette drive must match the type specified by the $DEVICE clause (an "automatic" determination of media types is not supported). Attempting to read or write a diskette which has been designated as the wrong media type results in an NPL error I93 - format error.

Programs which must access different media types may do so by trying to read sector zero, using each of the formats in turn, with appropriate error coding for recovery.

For example:

```
0010 $DEVICE(/D10)="/dev/fd048ds9"               :REM  try 360K format
  : S=1440                                       :REM 360K media size
  :DATALOAD BAT/D10,(0)A$
  : ERROR $DEVICE(/D10)="/dev/fd096ds15"         :REM 360K failed, try 1.2MB
  : S=4800                                       :REM 1.2MB media size
  : DATALOAD BAT/D10,(0)A$
  : ERROR $DEVICE(/D10)="/dev/fd096ds9"          :REM 1.2MB failed, try 720K
  : S=2880                                       :REM 720K media size
  : DATALOAD BAT/D10,(0)A$
  : ERROR PRINT "Cannot read diskette"
  : END
0020 PRINT S
  : REM At this pint we can access raw media in /D10. Media
  : REM size is S (256-byte) sectors.
```

HINT: It is recommended that the same NPL address (i.e. /D10) be used to access any one drive (e.g., drive 0), if access to different types is required. Attempting to define, for example, /D10 as "/dev/fd048ds9" and /D20 as "/dev/fd096ds15" at the same time may result in spurious errors on some systems.

5.2.4 "Raw" Diskettes

In the context of this manual, a "raw" diskette is one that does not contain a UNIX operating system file structure, but rather contains an NPL catalog and index as the primary file structure. The NPL RunTime treats the entire diskette as a diskimage file, and accesses sectors on the diskette directly, as opposed to accessing a diskimage file stored on the diskette using the UNIX file system.

Although "diskimage" files on diskettes can be used under UNIX, it would be necessary to mount each diskette as a UNIX file system before access to the diskimage file and to dismount the diskette after access.

Although the 360K, 720K, 1.2MB, 1.44MB, and 2.88MB diskette formats use 512-byte sectors, an NPL "sector" continues to refer to 256 bytes of information. All access to the diskettes (such as DATALOAD BA) determines the appropriate 512-byte sector containing the required information. The RunTime then performs a read of the 512-byte sector, modifies the appropriate 256-byte section and rewrites the full 512-byte sector. Consequently, the difference between diskettes, once they have been initialized (using $FORMAT DISK and SCRATCH DISK statements) is transparent to most applications, provided the $DEVICE specification contains the appropriate media type clause.

When raw diskettes of any class are formatted, bytes 3 and 4 of sector 0 are set to indicate the media size in 256 byte units (320K=1280 sectors, 360K=1440 sectors, 720K=2880 sectors, 1.2MB=4800 sectors, 1.44MB=5760 sectors, 2.88MB=11520 sectors) which may be used (after subtracting 1 sector) as the END= parameter to a subsequent SCRATCH DISK statement.

NOTE: **A value of zero was placed in these bytes by previous UNIX implementations of the RunTime.**

$FORMAT DISK is not supported on all UNIX hardware platforms.

Performance

Since the minimum amount that may be read from or written to a diskette is one sector, the access to the new types of media may be noticeably slower when writing single sectors (this entails reading a 512-byte sector, modifying the appropriate part and then rewriting it.) Since rewriting incurs an overhead of a complete rotation period, single sector writes can be expected to require an average of 1.5 rotational periods, compared to an average of .5 rotational periods for the 320K media (when supported by the operating system). Multi-sector writes (COPY/MOVE/VERIFY) do not incur this performance penalty, except possibly on the initial and final sectors of a multi-sector access. As a result, these operations (and reads of all types) perform at speeds that are approximately equivalent to the 320K media.

NOTE: **320K raw format is not supported by NPL under UNIX. The above information, on performance, is provided only as a reference for developers who use 320K raw access on other platforms.**

Media Defects

All types of raw diskettes used by NPL must be defect free. The $FORMAT DISK statement will report an error if media defects are detected during the format procedure.

NOTE: **The types of diskettes used for 360K media and 1.2MB (high density) diskettes are a different type. These types of diskettes are not interchangeable in any way. This is also true with the 720K, 1.44MB, and 2.88MB 3-1/2" diskettes in that the recommended media for each are different types.**

Drive/Media Compatibility

360K diskette drives are not capable of accessing 1.2MB media. 1.2MB diskette drives, however, are capable of reading 360K diskettes, if the $DEVICE statement points to the correct UNIX device name (e.g. /dev/fd048ds9).

NOTE Due to an inherent problem in 1.2MB drive technology, writing to 360K diskettes with a 1.2MB drive may produce a diskette which cannot be read on a 360K diskette drive.

The 3-1/2" 720K diskette drives are not capable of accessing 1.44MB media or 2.88MB media nor the 1.44MB drives of accessing 2.88MB media. 1.44MB and 2.88MB diskette drives, however, are capable of reading 720K diskettes, if the $DEVICE statement contains the correct UNIX device name (e.g. /dev/fd0135ds9). Similarly, 2.88MB diskette drives are capable of reading 1.44MB media if the $DEVICE statement contains the correct UNIX device name (e.g. /dev/fd0135ds18).

Data Integrity

A UNIX diskette cannot be read from or written to directly by NPL (unless the diskette contains a UNIX filesystem and is mounted on the main file system, in which case it can be accessed through the UNIX file system), and UNIX applications cannot read or write NPL raw diskettes.

Developers should take precautions to ensure that application programs do not modify diskettes unless they have been validated (e.g, locating an expected file by name).

WARNING--NPL does not attempt to prevent programs from modifying UNIX diskettes, which could result in corrupting the data on the diskette. Similarly, it is possible that UNIX applications could corrupt the NPL diskettes if direct access to the media is made.

Multi-User Access

Caution is recommended in shared access to "raw" diskettes. If two terminals are accessing diskettes without performing $OPEN statements, and multiple diskettes are being accessed, it is possible that UNIX buffering may result in data from a previous diskette being returned to a READ operation. Of course, programs allowing shared access to a diskette are likely to have problems in any environment.

$FORMAT DISK

The NPL statement $FORMAT DISK is not supported on all UNIX systems. As such, it is necessary to preformat diskettes under UNIX before using them with NPL. The following illustrates examples of formatting diskettes under UNIX.

```
format /dev/rfd048ds9           Formats media as 360K
format /dev/rfd096ds9           Formats media as 720K (5-1/4")
format /dev/rfd096ds15          Formats media as 1.2MB
format /dev/rfd0135ds9          Formats media as 720K (3-1/2")
format /dev/rfd0135ds18         Formats media as 1.44MB
format /dev/rfd0135ds36         Formats media as 2.88MB
```

HINT: If an application requires $FORMAT DISK be executed under program control, an alternative would be to replace the $FORMAT DISK commands with a $SHELL statement that executes one of the above UNIX command lines.

5.2.5 Error Handling

UNIX always displays any I/O type errors on the system console. Many NPL programs, including the NPL Utilities, are designed to capture (using ERROR statements) diskette type errors and handle them programmatically. However, it is recommended that users be advised to execute programs that access "raw" diskettes from the console only. This way the spurious UNIX error messages only affect the terminal using the diskettes.

5.2.6 Programming Considerations

There are several issues in the handling of "raw" diskettes under UNIX. Please review these carefully and make appropriate modifications to programs if required.

Mounting Diskettes

The UNIX operating system internally buffers data read from a diskette drive, and may continue to rely on this buffered data even in the event that the diskette drive door has been opened. The buffered data is not discarded until the device is logically closed and re-opened. Although NPL automatically logically OPENs a file that has been CLOSED, it cannot automatically perform a logical CLOSE on its own. Any NPL program using a device must logically CLOSE it whenever a new diskette is mounted. Programs should also CLOSE devices when they are finished so other programs or terminals may successfully access the respective drives.

To logically CLOSE a NPL device, either a $CLOSE statement or a $DEVICE statement must be executed. For example, assume the NPL disk address for the diskette is D10, and that D10 has been SELECTED as device #1 (SELECT #1/D10) in the internal NPL device equivalence table. Any of the following statements logically close the NPL device D10 and the corresponding UNIX special file names:

```
$CLOSE
$DEVICE(/D10)=$DEVICE(/D10)
$DEVICE(#1)=$DEVICE(#1)
```

I93 Error if No Diskette in Drive

The NPL error code returned to programs that attempt to access a diskette that is not mounted is an I93. This may cause some confusion for programs that expect an I93 error when a diskette is in the drive, but is not formatted. The NPL General Backup, Copy, and Restore Utilities are affected by this incompatibility.

NOTE: This incompatibility results from UNIX returning the same error code when a format error is encountered as when there is no diskette in the drive. There is no method by which a NPL program can differentiate between these two conditions.

5.3 Diskimage Files

All features of diskimage files, as described in Section 7.3 of the NPL Programmer's Guide, are fully supported under UNIX. A diskimage file can have any valid UNIX filename.

HINT: Developers are advised to consider the file naming restrictions on all NPL-supported platforms they wish their applications to run on when choosing names for diskimage files.

5.3.1 Naming Conventions

$DEVICE statements for diskimage files under UNIX should use the following form:

```
$DEVICE(/XXX)="[pathname/]diskimage"
```

where:

XXX	Any valid NPL disk device address
pathname	Any valid UNIX pathname (e.g., /usr/BASIC2C). If no pathname is specified, the current working directory is used.
diskimage	The name of the diskimage file.

HINT: It is recommended that the extension .BS2 be used for diskimage filenames to clearly distinguish diskimage files from other files stored in the UNIX file system.

NOTE: UNIX is case sensitive. Therefore, the upper and lower case of a $DEVICE statement is respected for pathnames and filenames.

HINT: It is recommended that only uppercase pathnames and filenames be used to provide compatibility with other NPL platforms which are case-insensitive.

5.3.2 File Allocation Considerations

The following file allocation issues should also be considered under UNIX.

SCRATCH DISK

Under UNIX, SCRATCH DISK statements directed against a diskimage file write to all sectors up to the specified end catalog area. Consequently, the SCRATCH DISK statement takes several seconds to execute. This is necessary to ensure that UNIX actually accesses the physical space required for the diskimage file.

Non-Catalogued Diskimage Space

Use of non-catalogued diskimage space under UNIX may cause difficulties. Use of a DATA SAVE statement beyond the end catalog of a given diskimage does not dynamically expand the physical size of the diskimage file. Only the individual sector written to is allocated to the file. Intervening space is not allocated until it is written to. Thus, it is possible that subsequent attempts to write to sectors between the end catalog and the physical end of file may fail if there is insufficient physical space to allocate on the disk at that time. Any attempts to read sectors beyond the end catalog that have not been written to results in the specified input variable being filled with HEX(00)s.

Programs relying on the use of non-catalogued disk space must be modified to write to all sectors from the end catalog to the logical end of diskimage before attempting to use non-catalogued disk space.

Altering Diskimage File Size

Under NPL, the physical size of diskimage files can be altered dynamically. One statement that does this is the MOVE END statement, executed against a disk address specified by the operator.

When MOVE END is used to expand a diskimage file, all new sectors are written to (as with SCRATCH DISK). However, when MOVE END is used to DECREASE the size of a diskimage file, physical space is not de-allocated. Only the catalog end parameter internal to the diskimage is affected. To actually release the physical space for reuse by other files requires a multiple step procedure:

1. MOVE END on old file to required size. This changes the catalog END, and does not actually perform a file space de-allocation.

2. SCRATCH DISK on new platter address with new size.
3. COPY or MOVE from old platter address to new platter.
4. SCRATCH old platter with new size.
5. COPY new platter onto old platter.
6. DELETE new platter (using the UNIX "rm" command).

HINT: Steps 4, 5 and 6 may be replaced by the UNIX mv command (e.g., mv newfile oldfile), which is much faster. This renames the new and deletes the old file.

NOTE: **If disk space is not sufficient to make a second copy on-line, then steps 2 and 3 must be replaced by a full disk backup procedure, and steps 5 and 6 must be replaced by a full disk recovery.**

5.3.3 Effect of Disk I/O on Task Interleave

All shared logic CPU's must have some mechanism of sharing time between different tasks operating on the system. Under UNIX, the time sharing method does not necessarily provide an even distribution of CPU time among different terminals on the system (unlike the Wang 2200 which provides a perfectly even distribution of CPU time among active terminals). Under certain circumstances, when several terminals are performing disk I/O, the time sharing between terminals may become highly skewed so that some terminals perform very well while others perform poorly. The NPL RunTime incorporates a time slice release option to help even out time sharing between all terminals.

The Timeslice Release Option

The NPL RunTime can be set to release the remaining time slice for a terminal following the release of a diskimage file. Byte 14 of $OPTIONS controls this feature. If byte 14 of $OPTIONS is set to HEX(00) (the default), no time slice release occurs. If byte 14 of $OPTIONS is set to HEX(01), the remaining time slice is released whenever a diskimage file lock is released.

For example:

```
10 DIM X$64          :REM ALWAYS USE A 64 BYTE VAR. WITH $OPTIONS
20 X$=$OPTIONS
30 STR(X$, 14, 1)=HEX(01)
40 $OPTIONS=X$
```

Puts the timeslice release option into effect. Refer to the NPL Statements Guide, $OP-TIONS, for more details.

Release of Diskimage File Lock

As stated, if the timeslice release option is specified, the timeslice release occurs whenever a diskimage file lock is released. Diskimage files are released under two circumstances:

- If a $OPEN directed to that diskimage has been executed, the diskimage is released only after a $CLOSE directed to that diskimage or by activity which causes the RunTime to logically close the diskimage (HELP or execution of $SHELL).
- If no $OPEN has been issued, the diskimage is released following completion of each disk I/O statement.

Performance Considerations

Although the timeslice release option improves the distribution of CPU time between terminals, it decreases performance for disk intensive applications. The following guidelines may help indicate what type of applications may benefit from the timeslice release option:

- Applications which perform intensive disk I/O operations and use $OPEN/$CLOSE on multiple terminals with no keyboard input will benefit.
- Applications which perform intensive disk I/O, but do not use $OPEN/$CLOSE experience the most significant decrease in performance. Here, the timeslice is released following every disk operation.
- Applications which are not disk intensive do not gain any benefit.

HINT: It is recommended that applications be tested in a multi-user environment before implementing the timeslice release option. If performance on different terminals varies widely, the timeslice release option can be implemented to see if any improvement occurs.

NOTE: Refer to Section 7.5.2 for details on the implementation of $BREAK on UNIX systems.

5.3.4 RAM Disks

A RAM disk is a device driver that provides a block interface to memory. A RAM disk can be used like any other block device, including making it into a file system using the mkfs command. Such a file system can be accessed by the application using traditional disk I/O statements. The advantage is that response time for disk operations directed to the RAM disk is much faster than for operations directed to a file system situated on the hard disk. The disadvantage of a RAM disk is that all data stored there is lost whenever the machine is turned off or when power is otherwise interrupted.

NOTE: RAM disks are implemented under SCO UNIX and Xenix, but may not be available on all UNIX platforms. Refer to your system documentation for more information.

Creating a RAM Disk

The following steps will create a RAM disk under SCO UNIX V/386:

1. Create the device that the RAM disk will reside on, using the mknod command. For example:

   ```
   mknod /dev/ram1 b 31 145
   ```

   creates a 64K permanent RAM disk (the size of the RAM disk can go up to 32MB).

2. Create a file system on the RAM disk, using mkfs command. For example:

   ```
   mkfs /dev/ram1 64
   ```

   creates a 64K file system.

3. Mount the selected device on a specified mount point, using the mount command. For example:

   ```
   mount /dev/ram1 /mnt
   ```

   mounts the RAM disk on /mnt.

Refer to the UNIX system documentation for further configuration details.

Accessing a RAM Disk

Accessing a RAM disk from NPL is very easy. Typically the only statements that need to be changed are the $DEVICE statements assigning the NPL disk address to the UNIX file. For example:

```
$DEVICE(/D21)="/mnt/PLATTER1.BS2"
```

Some additional logic may be needed to copy the NPL programs or work files into the RAM disk. This logic could be structured so that the first terminal to log in creates the proper files in the RAM disk. Subsequent terminals would then detect the existence of the files in the RAM disk and would skip the file creation step. The simplest technique involves having all the files you intend to place in the RAM disk in one or more separate diskimages on the hard disk and then copy the files to RAM disk during the start up routines.

NOTE: **Should the RAM disk not be available, program operation could still continue using the hard disk based files.**

The program that creates the diskimage file in the RAM disk should check for errors during file creation (insufficient space, for example) and remove any partial diskimage file created.

Use of RAM Disks to Functionally Replace Global Partitions

One of the more common uses of the global partition is to store record-locking information or other types of variable data that must be accessed by all terminals. For applications like this, RAM disks provide an excellent functional alternative to the global partition.

The basic technique for converting these applications to NPL is to use a disk file to store the common data that must be accessed by all terminals. This does require programming modifications, but these changes are generally not extensive.

The major concern with this type of application has been performance. The additional disk I/O required to read/write this common information has sometimes caused overall performance of the application to degrade. Use of a RAM disk to store the file containing the common information resolves all performance issues relating to this technique. Thus, when the RAM disk is used, the disk I/O used to read/write this common information now becomes memory transfer operations and performance is comparable to that of a global subroutine.

HINT: Applications that are converted to this technique work well on NPL platforms where RAM disks are not supported--they just will not work as fast.

5.3.5 UNIX umask and ulimit

All UNIX files have user access privileges assigned. The NPL RunTime program does not set specific user access privileges to any files created during its operations. Instead, these access privileges are determined by the umask setting of the individual user who creates the file. Improper access privileges could result in difficulties in accessing diskimage files. In addition, the UNIX operating system limits the largest sector location in a file that can be accessed during a write to that file by the ulimit system variable.

Improper umask values could result in problems with the NPL RunTime security. Refer to Appendix A for details.

umask

The value of umask is the system default. If there is a need to make files public, the start up shell script should be changed to set umask to a suitable value. Refer to Section 2.6.2 for details on setting the umask value.

ulimit

The ulimit value under UNIX is a process specific value which indicates the largest sector location that can be written to in any file (files of any size can be read). For example, the ulimit system variable might be set to only allow access of up to 16 MB of a file but an application needs to access a full 32 MB of a larger file.

With the need to access files larger than 16 MB for extended diskimage files, the NPL RunTime is required to set the ulimit to very large values. In addition, the original ulimit is restored before running any processes created by Invoke (! command) or $SHELL operations. This could produce problems if $SHELL statements are being used to work with large diskimage files (e.g., restore, copy, move, etc.). In general, such operations should only be carried out by procedures which have been granted super-user privileges and which specifically set ulimit to a sufficiently larger value.

5.3.6 Other Issues

There are some additional factors to be aware of in a UNIX environment when working with diskimage files.

- $DEVICE statements which reference directory paths or drive designations explicitly may need to be modified to include reference to the /usr (or other mounted file system) directory. Refer to this Chapter and $DEVICE in the NPL Statements Guide for details.

For example:

```
$DEVICE(/D11)="/usr/PROGS/ARPROGS.BS2"
$DEVICE(/D12)="/usr/DATA/ARMASTER.BS2"
```

- UNIX filenames are case sensitive. $DEVICE statements must be checked to ensure that the proper case is used.

- The use of diskimage files on diskettes is inconvenient under UNIX. UNIX requires that a removable disk device be mounted as a file system each time a new diskette is placed in the drive before normal disk access is permitted. Although diskettes can be logically mounted as a file system from UNIX, this is awkward from within an NPL program. This restriction does not apply for "raw" access to removable devices, such as when accessing "raw" 720K diskettes.

- Byte 39 of $OPTIONS may be used to specify that no implicit $OPEN is to be performed on DATA LOAD BA and DATA LOAD BM when the platter has not been explicitly hogged by $OPEN. Refer to Section 8.2.2 and Section 7.2 for details.

- No file locking is performed on Read Only files. Refer to Section 7.2 for details.

5.4 Supported Console Monitors and Controllers

NPL for UNIX supports monochrome and color controllers and monitors for the console terminal. Chapter 6 discusses the use of these controllers and monitors in detail.

5.4.1 File Naming Conventions

Under the UNIX implementation of NPL, the terminal type for an IBM-compatible PC monitor on the console terminal is determined automatically by the NPL RunTime program.

The monitor type for the console is stored in byte 3 of $MACHINE. The following are valid monitor types.

Monitor Type	Value of byte 3 of $MACHINE
ASCII	C
Monochrome	M
Color	C

The terminal type is stored in byte 9 of $MACHINE. The following are valid monitor types:

Terminal Type	Value of byte 9 of $MACHINE
HEX(00)	Wang PC
HEX(01)	Wang 2210a
HEX(02)	Altos III
HEX(03)	VT100/VT200, Bull HDS 3, NCR 4970
HEX(04)	IBM PC (using /R)
HEX(05)	Wyse 60,150,160
HEX(06)	Wyse 50, Bull HDS 1
HEX(07)	Wang 2x36DE/DW
HEX(08)	Altos V
HEX(0B)	Wang 370
HEX(0C)	Spectrix SPX 701
HEX(0D)	IBM 3151

The system console for an IBM AT (monochrome or color) is supported under NPL, but requires that the terminal type for NPL be set up as "imon". Since the UNIX terminal type for the console is "ansi", the BASIC2C_TERM variable should be set to "imon". To do this, add the following statement to the .profile file:

```
if [ $TERM='ansi' ]
then
    BASIC2C_TERM=imon
    export BASIC2C_TERM
fi
```

Refer to Section 6.4 for more information.

5.4.2 Using Emulation Products

Many emulation products allow the use of IBM compatible PCs as terminals for UNIX systems.

NOTE: Niakwa does not officially support the use of any emulation product. However, Niakwa recommends the use of ANSI emulation modes (which supports local printing in ANSI mode (imon)).

A $OPTIONS byte is available to provide better results with use with emulation products. The bytes are described below:

Byte 31 of $OPTIONS

This byte controls certain features for terminal emulators which do not provide 100% support of the terminal being emulated. Refer to the NPL Statements Guide, $OPTIONS, for more information.

5.5 Printers

NPL under UNIX provides support for print output to parallel ports, serial ports, local printers attached to supported terminals, or UNIX text files.

NPL permits the automatic translation of characters as they are sent to a printer (or spooler) using a lookup table. This is performed using the Printer Translation Table option. For a full discussion of this facility, refer to Chapters 7 and 13 of the NPL Programmer's Guide.

If no translation is performed, all characters sent by the NPL program are passed directly to the specified print device with no modification. Thus, all printer control sequences are the responsibility of the programmer.

5.5.1 On-Line Printing

Either parallel or serial system printers can be accessed under UNIX. Printing performed to printers which do not have the hardware/software capabilities of setting auto line feed attributes should be done on-line or local, so that output is properly formatted. In addition, should a program attempt to send control sequences to the print device, on-line or local printing may also alleviate the problem of control code stripping that may occur within the UNIX systems spooling facility.

Parallel Device Equivalences

Different UNIX platforms have specific device names to access device resources, especially the parallel printer port. The following device equivalences are used for parallel printing under SCO System V UNIX, SCO Xenix, and Interactive UNIX:

```
$DEVICE(XXX)="/dev/lpxx"
```

where lpxx is defined as one of the following device names:

```
lp0              main parallel port
lp0i
lp1              parallel port on monochrome card
lp1i
lp2              alternate parallel port
lp2i
```

The "i" following each device will cause the printer device to send an "init" each time the device is opened. This "init" will reinitialize the port canceling the current printer process. For more details, refer to the operating system documentation.

The following device equivalence is used for parallel printing under Altos UNIX:

```
$DEVICE(XXX)="/dev/plp"
```

Serial Device Equivalences

The following device equivalence is used for serial printing under all NPL-supported Intel UNIX platforms:

```
$DEVICE(XXX)="/dev/ttyxx"
```

where ttyxx is the port number of the serial port to which the printer is attached.

NOTE: The device /dev/lp is the default for printer addresses /015 and /004. This may be changed to an appropriate device name for the UNIX system being used. Alternately, an existing printer device can be linked to the name /dev/lp (e.g. "ln /dev/lp0 /dev/lp", or "ln /dev/tty01 /dev/lp", etc.).

Line Feed (ALF) Handling

NPL automatically generates a line feed after a carriage return in all output directed to printer devices. This line feed can be suppressed by the ALF option in the device equivalence statement. For example:

```
$DEVICE(/215)="/dev/lp0 ALF=x"
```

where x is either Y (Yes) or N (No).

The designation of ALF=N suppresses the automatic line feeds for the selected 215 printer address. The designation of ALF=Y does not suppress the automatic line feeds for the selected 215 printer address. If ALF is not specified, the default value of "Y" is used.

$OPEN/$CLOSE Considerations

$OPEN and $CLOSE should be used when directing output to on-line print devices. Refer to Section 5.5.6 for more details.

Printer Not Ready

Printer not ready signals can be caused by one of two conditions:

- Print device is attached but not in ready mode (i.e., someone else is using it with a valid $OPEN, or printer is deselected).
- Print device is not attached and has been selected erroneously.

If using a Parallel printer, the following occurs:

- The terminal hangs until the printer is ready. If this situation occurs, the condition must be corrected before the terminal can resume processing.
- If no device is attached the output and the terminal continue to hang. If this condition cannot be corrected, it is necessary to kill the task of the hung terminal. Refer to Section 5.5.7 for details.

If using a Serial printer, the following occurs:

- The terminal hangs until the printer is ready. If this situation occurs, the condition must be corrected before the terminal can resume processing.
- If no device is attached, the output continues as if a printer was attached, however, all data is discarded.

5.5.2 Printing Using the UNIX Print Spooler

UNIX has a built in print spooling capability. Niakwa recommends using this capability in multi-user environments which have both NPL and non-NPL applications running concurrently to avoid intermingled printed output.

NOTE: Some flavors of UNIX provide support for local terminal printing through the use of the UNIX print spooler. This provides an alternative to the NPL local printing options that may be considered. Refer to the UNIX documentation for more information.

Device Equivalences

Print spooling on Intel UNIX platforms is handled by a print spooler program. The name of the print spooler program and its available command line options vary between different UNIX platforms.

The following device equivalence statements accesses spooled printing on an SCO Xenix system:

```
    $DEVICE(/XXX)="(lpr -s -ddest)"
```

or

```
    $DEVICE(/XXX)="(lpr -s)"
```

In the above examples, the -s silences the spooler from printing a "Request id" message when a printer request is made. The "-ddest" specifies a particular destination for output, either a printer name or printer class, which can be a parallel or serial printer. The second example assumes that output is directed to the default system destination.

The following device equivalence statements accesses spooled printing on SCO System V UNIX, Interactive UNIX, or Altos System V UNIX:

```
    $DEVICE(/XXX)="(lp -s -ddest)"
```

or

```
    $DEVICE(/XXX)="(lp -s)"
```

The -s and -d options function identically to those under SCO Xenix in the previous example.

The following device equivalence statements also accesses spooled printing under Altos System V UNIX:

```
    $DEVICE(/XXX)="(lpr -px)"
```

or

```
    $DEVICE(/XXX)="(lpr)"
```

In the first example, the -p option directs the output to a specific printer, where x is the printer number assigned by the Altos pconfig program. The second example directs output to the default system destination.

Line Feed (ALF) Handling

The NPL RunTime program automatically generates a line feed after a carriage return in all output directed to printer devices. This line feed can be suppressed by the ALF option in the device equivalence statement. For example:

```
    $DEVICE(/215)="(lp -s) ALF=x"
```

where x is either Y (Yes) or N (No)

The designation of ALF=N suppresses the automatic line feeds for the selected 215 printer address. The designation of ALF=Y does not suppress the automatic line feeds for the selected 215 printer address. If ALF is not specified, the default value of "Y" is used.

$OPEN/$CLOSE

A $OPEN statement always succeeds when directed to the UNIX print spooler.

Printer Not Ready

Any attempt to print to the UNIX print spooler succeeds providing the spooler has been properly configured. A "printer not ready" condition should not occur.

Other Issues

The following issues should also be considered:

- Ordinarily, the use of the print spooling process works with most applications. However, on some UNIX systems applications which use control code sequences for printing, such as for use with graphics data, may find that spooling strips out some of those control sequences. Therefore, programs of this type may have problems using the UNIX print spooler.

- A $CLOSE statement directed against a printer device defined as the print spooler will halt the sending of the current file and release it to control of the despooler for printing. In addition, any other action that closes the print file has this same effect. This includes exiting from the NPL RunTime and pressing HELP.

5.5.3 Terminal Printers (Local Printing)

Many of the terminals supported by NPL under UNIX can support a serial or parallel printer attached to the AUX port of the terminal. This is a particular advantage when using the terminal on a remote basis.

NOTE: Some flavors of UNIX provide support for local terminal printing through the use of the UNIX print spooler. This provides an alternative to the NPL local printing options that may be considered. Refer to the UNIX documentation for more information.

Device Equivalences

Terminal printers differ significantly from terminal printers on the Wang 2200 in that no special NPL address is required.

The device equivalent for local printers is:

```
$DEVICE(xxx)="/dev/tty LCL=Y"
```

where xxx is a valid NPL printer address (/204 for example).

Refer to Section 6.2.5 in this Supplement for details on configuring local terminal printers.

Line Feed (ALF) Handling

The NPL RunTime program automatically generates a line feed after a carriage return in all output directed to printer devices. This line feed can be suppressed by the ALF option in the device equivalence statement.

For example:

```
$DEVICE(/215)="/dev/tty LCL=Y ALF=x"
```

where x is either Y (Yes) or N (No)

The designation of ALF=N suppresses the automatic line feeds for the selected 215 printer address. The designation of ALF=Y does not suppress the automatic line feeds for the selected 215 printer address. If ALF is not specified, the default value of "Y" is used.

$OPEN/$CLOSE

Use of $OPEN/$CLOSE is optional when output is directed to a local printer, since only the terminal that is attached to the printer may send output to it. $OPEN always succeeds when directed to a local printer. Refer to Section 5.5.6 for more details on the use of the $OPEN and $CLOSE statements.

Printer Not Ready

Printer not ready signals can be caused by one of two conditions:

- Printer is attached but not in ready mode.

- Print device is not attached and has been selected erroneously.

When using a local printer, the following occurs:

- The terminal hangs until the printer is ready. If this situation occurs, the condition must be corrected before the terminal can resume processing.
- If no device is attached the output and the terminal may hang, if this condition cannot be corrected, it is then necessary to kill the task at the hung terminal. Refer to Section 5.5.7 for details.
- If no device is attached the output may continue as if a printer was attached, however, all data is discarded.

The exact behavior depends on the type of terminal in use and may also depend on the specific printer in use. Refer to the terminal and printer user's manual for specific information.

5.5.4 Text Files

Print output may be directed to standard text files by specifying the name of the file as a device equivalence.

Device Equivalences

Print output can be directed to a text file by specifying the name of the file as a device equivalence to any valid NPL printer address.

An example of a device equivalent for a text file is:

```
$DEVICE(xxx)="/usr/BASIC2C/test.dat"
```

where xxx is a valid NPL printer address.

If a file test.dat does not exist, UNIX automatically creates it (remember that UNIX is case sensitive).

Line Feed (ALF) Handling

Output directed to ASCII text files do not have a carriage return inserted by the NPL Run-Time program before each line feed. This makes the text file produced accessible to UNIX utilities including the print spooler. This may be overridden by the ALF=N option if output to the file contains control codes (HEX(0D)), (such as graphic data). If control codes are needed, the NPL program must provide line feeds, (HEX(0A)), either explicitly or by SELECT PRINT 0XX.

$OPEN/$CLOSE Considerations

$OPEN and $CLOSE should be used when directed to the text file. If a proper $OPEN is not executed, the possibility of reports being intermingled when two or more users access the file simultaneously may occur. Refer to Section 5.5.6 for more details on the use of the $OPEN and $CLOSE statements.

Printer Not Ready

When printing to a text file, a printer not ready condition should not occur if the file is defined properly in the $DEVICE statement. If the device is not properly defined, a P48 - Illegal Device Specification error is generated.

5.5.5 Line Feed Handling

UNIX expects printers to automatically perform a carriage return (HEX 0D) whenever a line feed (HEX 0A) is encountered. NPL applications normally expect a printer to do a line feed (HEX(0A) whenever a carriage return (HEX(0D) is encountered. NPL handles this by replacing all carriage returns with either a CR/LF sequence (if printing to an on-line device or local printer) or a LF (if printing to a spool file or another non-device).

The ALF (Auto Line Feed) Option

Special handling of carriage returns by the NPL RunTime program can be suppressed by the operator at execution time from the NPL HELP processor (PRINTER CONTROL selection, setting AUTO-LF OFF).

Alternatively, the Auto Line Feed option may be directly controlled by a NPL program. This is accomplished by using a $DEVICE statement for printer type devices.

The form of the ALF option may be set to "Y" or "N". A "Y" indicates that the special output options ARE used, while an "N" indicates that the special output options are not to be used. For example:

```
    $DEVICE(/215)="/dev/lp0 ALF=N"
```

suppresses special handling of carriage returns for the selected 215 printer address. If ALF is not specified, the default of "N" is used.

5.5.6 $OPEN/$CLOSE Considerations

$OPEN and $CLOSE should be used when directed to the actual on-line print devices (refer to the NPL Statements Guide for details on the use of the $OPEN and $CLOSE statements). However, there are several points to consider.

- Printer hogging is only enforced by the NPL RunTime program (both the Interpretive and Non-Interpretive). Printing from terminals running native UNIX applications while printing from the NPL RunTime program may result in output from both programs printing simultaneously. For this reason, if other UNIX applications are being used on the system, it is a good idea to use the UNIX print spooler, if possible.

- Printing to a printer without executing a $OPEN implicitly hogs the printer as though a $OPEN statement was executed. This printer is released upon any of the following:

  - Executing a $CLOSE statement.

  - Executing a LOAD RUN statement.

  - Exiting from the NPL RunTime program.

  - Invoking the NPL HELP processor.

5.5.7 Killing Output to a Non-Existent Device

UNIX does not provide a mechanism for program detection of a printer not ready status when printing directly to UNIX print devices. Therefore, any attempt to print directly to a UNIX print device which is not ready results in the terminal hanging until the printer is ready (this includes printer ready checks).

If this situation occurs, it is necessary to correct the problem with the printer before the terminal can resume processing. If the printer condition cannot be corrected, it will be necessary to kill the task at the hung terminal.

To kill the task:

1. Login to another terminal as "root"

2. Enter:

   ```
   ps - ef
   ```

   This will list all processes currently in operation. The TTY column lists the controlling terminal. The numbers shown under the PID and PPID columns are the process ID and parent process ID for each unique task.

3. Enter:

   ```
   kill PID
   ```

   Where PID is the process ID number determined from the ps command.

4. If this fails, enter:

   ```
   kill PPID
   ```

   Where PPID is the parent process ID number determined from the ps command.

WARNING--Killing a process in this manner may have an adverse affect on the NPL application if a critical update was being performed.

5.5.8 Printer Translation

NPL permits the automatic translation of characters (from the XLA=Y device clause) as they are sent to a printer (or spool file) using a look up table. This is performed from the NPL Printer Translation Table option. For a full discussion on printer translation and the XLA=Y device clause, refer to Section 7.8.7 of the NPL Programmer's Guide.

If no translation is performed, all characters sent by the NPL program are passed directly to the specified print device with no modification. Thus, all printer control sequences are still the responsibility of the programmer.

5.6 Mouse Support

A mouse device is not supported under the UNIX implementation of NPL.

5.7 Serial Devices

The RunTime contains limited ability to read and write raw data from a serial port using the C620 $GIO microcommand. The following section discusses the use of serial devices under UNIX for NPL.

5.7.1 Naming Conventions

Serial ports under UNIX are accessed as a raw serial device through the device designation /dev/ttyxx in the $DEVICE statement, where xx is the logical port number.

NOTE: **The port being accessed must not be enabled as a terminal port if it is to be used for raw serial I/O operations.**

For example:

```
$DEVICE(/01C)="/dev/tty05"
```

or

```
$DEVICE(/01C)="/dev/tty05 TMO=Y"
```

directs subsequent I/O operations directed to address /01C to the port tty05. The optional designation TMO=Y instructs input operations directed to the serial port to return to program control without waiting for an end-of-record indicator to be present. Refer to Section 5.7.3 and Chapter 7 of the NPL Programmer's Guide for more details on accessing serial ports under NPL.

5.7.2 Sending Output to a Serial Port

NPL PRINT statements may be used to direct output to a serial port. In addition, the $GIO output microcommands, both single character and multi-character, may be used.

NOTE: **Output to a serial port is treated similarly to printer output: automatic line feed insertion and character translation are in effect unless overridden by the appropriate options of $DEVICE (ALF and XLA, respectively). Refer to Section 7.9 of the NPL Programmer's Guide for more details on accessing serial ports under NPL.**

5.7.3 Input from a Serial Port

NOTE: The following discussion refers to functionality when the TMO=Y option has been specified in the $DEVICE statement, as explained above.

The UNIX version of NPL contains limited support for accepting input from a serial port. This support is intended to provide a mechanism for using serial input devices where the interface requirements are very simple and straightforward. Applications which require the more sophisticated features of the Wang 2227 board or the Niakwa Scientific and Communications package are not supported under NPL for UNIX.

The C620 $GIO micro-command accesses binary data in the buffer for the port specified and returns as many bytes of data as are available or zero bytes, if no data is available.

For example:

```
10 DIM L$10,A$80                                 :REM GIO control, buffer
     : $DEVICE(/01C)="/dev/tty10 TMO=Y"          :REM read from port 10
20 $GIO/01C(C620,L$)A$                           :REM look for data on port
     : L=VAL(STR(L$,9),2)                        :REM get # of bytes received
     : IF L=O THEN $BREAK                        :REM check no data available
     : IF L0 THEN GOSUB XXX                      :REM goto subroutine xxx
     : GOTO 20                                   :REM process data
```

In this example, the first L bytes of A$ contain the data returned from the port.

This method of accessing data as input on the serial port is very primitive and has several restrictions that the programmer must consider:

- Communication parameters for the port must be established externally to NPL. The UNIX command stty may be used to define the baud rate, number of data bits, stop bits, and parity for serial ports.
- There is no error detection.
- There is no method for checking signals on-line (i.e., device not attached or not ready).
- None of the $GIO micro-commands supported by the NPL 2227 emulation driver are supported. Program modifications are required to support input from serial devices on the UNIX version of NPL.

- The input buffer for serial ports under UNIX is limited in size. If the buffer overflows, incoming characters are lost. However, under UNIX, XON/XOFF can be enabled to ensure that the input does not overrun the input buffer. XON/XOFF is user enabled by the UNIX stty command.

NOTE: **The size of the character buffer is user configurable. The buffer size is defined by a UNIX system parameter called NCLIST. Refer to your UNIX documentation for more details on the UNIX parameter NCLIST. In addition, refer to the hardware documentation of the serial board being used for further information on supported communication parameters.**

5.7.4 Asynchronous Serial Communications

When using the limited serial communication option of NPL under UNIX, the parameters of the tty port (baud rate, I/O control flags, etc.) are normally set to the parameters expected by the application. This is usually accomplished with the stty command. For these new parameters to be maintained, the UNIX tty ports must be kept open by a process. This is not a problem for ports that have been enabled and are continually being polled for login, but NPL communications require a disabled port. For an idle, disabled port, stty sets the new parameters momentarily and the parameters then revert back to their default states when the termination of the stty command closes the port.

To work around this problem, the parameters of both the tty port and a background process to keep the port open must be established. When executed, the following script file, "openport", will hold a port open indefinitely. This script file can be entered using any text editor. It should be stored in /usr/bin and given execute privileges (chmod a+x /usr/bin/openport).

```
#   @(#)openport
#
         while true ; do
              sleep 3600
         done
#
#   KEEPS A PORT OPEN BY PUTTING IT TO SLEEP FOR 1 HOUR
#   INTERVALS.
```

The following command line can be inserted into the user's .profile or /etc/rc file to set the parameters of a tty port and keep it open by executing the above shell script from /usr/bin/openport, as shown below.

```
(stty 2400 ixon ixoff -ixany ; /usr/bin/openport) /dev/tty1a &
```

NOTE: The above example is for setting the parameters for port tty1a and assumes the port is disabled. If executed from the user's .profile, the port will be closed when the user logs off. When executed from the /etc/rc file, the port will remain open until the process is killed or the system is shutdown.

5.8 The Tape Drive

NPL does not directly support access to a tape drive. However, all NPL diskimage files (e.g, PLATTER1.BS2) may be backed up using the standard UNIX "tar" utility. The $SHELL command can be used to access this method to perform a backup during program operation.

For example:

```
$SHELL "tar cvbf 126 /dev/rct filenames"
```

Where the filenames are the UNIX designations for the files or directories to be backed up to tape (the device name (/dev/rct) and blocking factor (126) used for the tape drive may vary from one system to the next).

5.9 Math Co-Processor Support

Use of the 80x87 math co-processor support is enabled by setting byte 16 of the $OPTIONS system variable. Valid values for the $OPTIONS byte include:

HEX(00)	Do not use math co-processor even if available.
HEX(01)	Use math co-processor for transcendentals if available.

Other values are reserved and should not be assigned to this byte.

Applications which use the math co-processor should set byte 16 to HEX(01) or they may experience a decrease in performance, since the RunTime default is not to use the co-processor.

For example:

```
0010 DIM X$64
  : X$=$OPTIONS
  : STR(X$,16,1)=BIN(1)
  : $OPTIONS=X$
```

NOTE: The $MACHINE system variable indicates whether a math co-processor is available. Byte 10 of $MACHINE contains the following values:

HEX(00)	No co-processor available.
HEX(01)	80x87-class co-processor available.

Developers making use of the math co-processor should be aware that there may be differences in precision of results and range of functional domain to some functions. In particular, the 80x87-class co-processors are generally accurate with 48-bit precision and have an exponent of 2 in the range +/- 16383. This does not normally present a problem, except where results or arguments approach overflow or underflow values.

5.10 Default Device Equivalences

When the NPL RunTime program is started under UNIX, the Device Equivalence Table contains the following default entries:

2200 Address	UNIX Equivalent
/B10 or /D10	/dev/fd096ds15
/310 or /D11	PLATTER1.BS2 in the current directory
/D12	PLATTER2.BS2 in the current directory
/215	/dev/lp
/204	/dev/lp

"Current Directory" is the directory selected at the time the RunTime is started. No entry is required for the keyboard or CRT screen. These are defaulted (/X01 and /X05, where X= 0 or 2) by the RunTime.

CHAPTER 6

TERMINAL CHARACTERISTICS

6.1 Overview

This chapter discusses the terminals supported by NPL on approved UNIX systems. This chapter also discusses the general operating system considerations regarding terminals supported by NPL under UNIX.

Section 6.2 discusses operating system characteristics.

Section 6.3 discusses supported terminals.

Section 6.4 discusses the console terminal characteristics using a monochrome controller.

Section 6.5 discusses the console terminal characteristics using a color controller.

Section 6.6 discusses color support under NPL.

Section 6.7 discusses graphics support with NPL under UNIX.

Section 6.8 discusses keyboard characteristics.

6.2 Operating System Characteristics

The following section discusses the characteristics of the UNIX operating system which affect terminal support. Among the topics discussed are how the terminal type is determined by the RunTime and where the terminal files are to be found. The HALT key function, local printer support and terminal configuration requirements are also discussed.

6.2.1 Determination of Terminal Type

NPL under UNIX uses the system variable BASIC2C_TERM to determine the terminal type. If BASIC2C_TERM is not defined, the UNIX TERM variable is used.

NOTE: The value for TERM is established at login time using the /etc/ttytype file.

BASIC2C_TERM should be used whenever the value of TERM does not correspond exactly with the terminal type identification expected by NPL. Refer to Section 6.3 for a list of supported terminals and the corresponding terminal types expected by NPL.

BASIC2C_TERM may be set and exported from the UNIX command line or from within the user's .profile file. The following example demonstrates setting BASIC2C_TERM from within the .profile file (this is the recommended method):

```
#  Set BASIC2C_TERM to Wyse 50 Terminal
#  export variable to the system
#  Redefine interrupt key to CONTROL/C
#
if [ $TERM = wyse50 ]
then
    BASIC2C_TERM=wy50
    export BASIC2C_TERM
    stty intr '^C'
fi
#
#  Sets BASIC2C_TERM to VT200 Terminal
#  export variable to the system
#  Redefines interrupt key to CONTROL/C
#  Download VT200 font files
#
if [ $TERM = vt200 ]
then
    BASIC2C_TERM=vt200
    export BASIC2C_TERM
    STTY INTR '^C'
    cat /usr/BASIC2C/VTFONT.VT200
    cat /usr/BASIC2C/VTKEYS.VT200
fi
```

NOTE: If BASIC2C_TERM is not defined for a given process, the UNIX TERM variable is used.

The default interrupt key under most UNIX environments is generally the "DEL" key. This may cause conflicts with some NPL RunTime applications, therefore the above example redefines the interrupt key to CTRL-C.

The above lines that begin with a # are comment lines and are ignored by UNIX.

6.2.2 Location of Terminal Files

NPL searches for the screen translation table file (SCREEN.xxxxx) and keyboard translation table file (KEYBOARD.xxxxx) first in the current directory, then in the directory set by the environment variable NIAKWA_RUNTIME, and finally in the default /usr/BASIC2C directory.

Refer to Section 6.3 for a list of the SCREEN and KEYBOARD files associated with each supported terminal. Refer to the NPL Statements Guide for details on using the $KEYBOARD and $SCREEN system variables. Refer to Sections 13.14 and 13.15 of the NPL Programmer's Guide for details on the use of the NPL Utilities used to modify the SCREEN and KEYBOARD tables and the FONT files.

6.2.3 Downloading FONT and KEYS Files

Several of the supported terminals under UNIX have screen character sets which support the use of "downloadable" fonts. In addition, the VT220 series and the Wyse 370 supports the use of a "downloadable" KEYS file. It is necessary to use the appropriate FONT and KEYS files for terminals that support downloadable files.

NOTE: **For the SCREEN.xxxxx and KEYBOARD.xxxx files to work correctly, the appropriate FONT and KEYS files must be downloaded.**

The appropriate files can be downloaded to the terminal being used by use of the UNIX "cat" command.

For example, to download the Wyse 60 font file, enter the following from the UNIX command prompt at a Wyse 60 terminal:

```
cat /usr/BASIC2C/WYFONT.WY60
```

Refer to Section 6.3 for a list of the FONT and KEYS files associated with each NPL-supported terminal.

NOTE: **The Wyse 150 and 160 do not support application key mode like the Wyse 60. As such, the file WYKEYS.ON must be downloaded on both the Wyse 150 and 160 prior to starting the RunTime. To return the keyboard back to its standard mode, the file WYKEYS.OFF must be downloaded on both the Wyse 150 and 160 prior to returning to the operating system. Refer to Appendix D of the NPL Programmer's Guide for more information.**

6.2.4 HALT Key Functionality

Under UNIX, the "HALT" key is the native "intr" key (usually the DEL key). This can be configured with the "stty intr" command. Niakwa recommends the use of the CONTROL-C as the "intr" key. This key is implemented by placing the following command in each user's .profile file:

```
stty intr '^C'
```

Refer to Section 2.3.2 for details on the .profile file.

6.2.5 Local Printer Support

Local printers may be accessed under NPL by using a special device designation "/dev/tty LCL=Y".

For example:

```
$DEVICE(/204)="/dev/tty LCL=Y"
```

directs all print output sent to the NPL print address /204 to the local printer port on the terminal. Both serial and parallel local printers are supported. Refer to Appendix D of the NPL Programmer's Guide for specific features of each supported terminal.

NOTE: Local printers are accessed only by the terminal to which they are attached.

Support for local printing under ANSI terminals using "imon" is supported. The use of the LCL=Y option on system consoles will be ignored.

6.2.6 Terminal Configuration Requirements

Under UNIX, supported terminals must be set for 8 data bits and XON/XOFF. There are no special requirements for baud rate, stop bits or parity, but the terminal setup must match the UNIX configuration for the serial port. Refer to Appendix D of the NPL Programmer's Guide for the suggested configuration for each supported terminal.

NOTE: UNIX serial ports support communication rates up to 38400 baud. However, baud rates higher than 9600 may have restricted cable length requirements.

HINT: If a terminal is experiencing intermittent flow control problems, it is recommended that the "ixany" parameter be turned off. To turn off the "ixany" parameter, enter the following from a UNIX prompt:

```
stty -ixany
```

For details on the stty parameters refer to the Commands Reference manual for your UNIX operating system.

NOTE: Baud rate may be limited by the terminal which is in use. Refer to Appendix D of the NPL Programmer's Guide for details on maximum communication rates for each supported terminal.

6.3 Supported Terminals

The following section discusses the terminals supported under NPL for UNIX.

6.3.1 NPL Supported Terminals

The following table lists the terminals that NPL supports under UNIX. This table also contains the "terminal kind" name which should be assigned to the BASIC2C_TERM variable and the names of the auxiliary files supplied for each terminal. Refer to Appendix D of the NPL Programmer's Guide for a complete description of terminal characteristics for all of the supported terminals.

Terminal Name	"Terminal kind"	Keyboard File Name	Screen File Name	Font File Name	KEYS File Name
Altos III	altos3	KEYBOARD.altos	SCREEN.altos3	N/A	N/A
Altos V	al5	KEYBOARD.al5	SCREEN.al5	VTFONT.al5	N/A
Bull HDS 1	wy50	KEYBOARD.wy50	SCREEN.wy50	N/A	N/A
Bull HDS 3	vt200 or vt220	KEYBOARD.vt200 or KEYBOARD.vt220	SCREEN.vt200 or SCREEN.vt220	VTFONT.vt200	VTKEYS.vt200
DEC VT100	vt100	KEYBOARD.vt100	SCREEN.vt100	N/A	N/A
DEC VT200 Series	vt200 or vt220	KEYBOARD.vt200 or KEYBOARD.vt220	SCREEN.vt200 or SCREEN.vt220	VTFONT.vt200	VTKEYS.vt200
IBM Console Monitor	imon	KEYBOARD.imon	SCREEN.imon	N/A	N/A
IBM 3151	i3151	KEYBOARD.i3151	SCREEN.i3151	N/A	N/A
NCR 4970	vt200 or vt220	KEYBOARD.vt200 or KEYBOARD.vt220	SCREEN.vt200 or SCREEN.vt220	VTFONT.vt200	VTKEYS.vt200
Spectrix SPX 701	vt100	KEYBOARD.vt100	SCREEN.vt100	N/A	N/A
Wang 2110A	w2110	KEYBOARD.w2110	SCREEN.w2110	N/A	N/A
Wang 2236 DE/DW	w2236	KEYBOARD.w2236	SCREEN.w2236	N/A	N/A
Wyse 50	wy50	KEYBOARD.wy50	SCREEN.wy50	N/A	N/A

Terminal Name	**"Terminal kind"**	**Keyboard File Name**	**Screen File Name**	**Font File Name**	**KEYS File Name**
Wyse 60,150, 160	wy60	KEYBOARD.wy60 KEYBOARD.wpc	SCREEN.wy60	WYFONT.wy60	N/A
Wyse 370	wy370	KEYBOARD.wy370	SCREEN.wy370	W370FONT.80 WY370FONT.132	WYKEYS.wy370

The Wyse 60, 150, and 160 support several different kinds of keyboards. Niakwa provides support for the ASCII and PC styles only. the file KEYBOARD.WY6 contains the default keyboard translation for the ASCII style keyboard. KEYBOARD.WPC contains the default translation for the PC style keyboard.

The PC style keyboard, KEYBOARD.WPC, if used on the Wyse 60, 150, or 160 terminals, must be renamed to KEYBOARD.WY6 before entering the RunTime.

6.3.2 Supported Terminal Characteristics

This section lists in a tabular format the features of the various terminals and console monitors supported by NPL on Intel UNIX platforms.

Terminal Characteristics

6-7 of SuperDOS to be pasted here

Terminal Features Notes

On terminals that support a single attribute, this attribute will be used for any NPL attribute or combination of attributes.

Bright, Blink, Reverse, and Underline attributes are supported in any combination. Certain restrictions may apply.

For terminals that support downloadable fonts, Niakwa provides a font file that provides complete emulation of the standard NPL character set. These font files may be modified by the developer. On terminals without downloadable fonts, the full NPL character set cannot be emulated. Typically, pixel graphics (HEX(C0) through HEX(FF)) and special characters may not be available.

The above terminal features chart is intended as an aid in determining the capabilities of a particular terminal within the NPL environment.

NOTE: **Limitations may exist that are unique from one terminal to the next. Developers should refer to the appropriate terminal's documentation for details on any restrictions that may apply.**

Console Characteristics

Controller	**Monitor**	**Mode**	**Box Type**	**Blink**	**Bright**	**Reverse**	**Underline**	**Color**	**Columns**	**Fonts**	**Alternate Character Set**
Any	Mono	Char	Char	Yes	Yes	Yes*	Yes	No	Only 80	No	No
Any	Color	Char	Char	Yes	Yes	Yes*	Yes**	Yes	Only 80	No	No

Console Notes

* Reverse + any other attribute used jointly appears as reverse video On Altos System V UNIX, SCO Xenix V, SCO System V UNIX, and Interactive UNIX.

** Underline is not supported on color consoles under XENIX/UNIX. On SCO UNIX and AT&T UNIX the control sequence for "underline" is ignored on color consoles. However, on Interactive UNIX, the "underline" control sequence generates a bright white foreground on red background combination. This color combination cannot be changed by the application.

6.3.3 Wang 2x36 Terminal Support

UNIX supports the use of the Wang 2x36 terminals on Niakwa approved hardware as supported by Digiboard. However, use of this terminal requires execution of a special program to set up proper XON/XOFF codes. If this program is not executed, flow control problems will occur and the terminal is not usable.

There are two methods that support the Wang 2x36 flow control under Intel UNIX based systems: directly from the operating system or through add on hardware and drivers. These options are discussed below.

Altos UNIX

Some older Altos UNIX based systems provided Wang 2x36 flow control through the operating system. For these systems, refer to the NPL Release III Altos UNIX Addendum or contact Niakwa's technical support.

DigiBoard Support

Wang 2236 DE/DW terminals may be used in conjunction with DigiBoard's DigiCHANNEL C/X Multiterminal controller system, using the "2200flow" option in DigiBoard's Ditty driver. 2200flow must be enabled when configuring 2236 terminals for operation; refer to the "Setting Terminal Options with Ditty" chapter in the DigiBoard Software Installation and Operation Manual for detailed instructions.

All terminal features discussed in Appendix D of the NPL Programmer's Guide are supported under UNIX with the following exceptions:

1. Terminal flow control settings should be set to: 19200 baud or lower, 8 bit transfer, 1 stop bit, and no parity.

2. The default stty value ^@ of the "swtch" and "susp" parameters must be remapped since ^@ is the default value generated by FN 0 on a 2236 terminal. For example:

   ```
   stty swtch '^[' susp '^['
   ```

3. The Wang 2236 HALT key is not supported with 2200flow, but is it mappable to another key (e.g., CTRL-S).

6.3.4 Use with Native Operating System and Functions

Although all the terminals listed in this chapter operate properly with NPL, some do not fully support UNIX operating system functions and utilities, and are therefore not well suited for use with native operating system features and utilities. Below is a summary of the terminals which are well suited for native operating system use, and those that are not well suited.

Terminals that are supported and well suited for use with the UNIX operating system functions include:

IBM AT Console
Altos III
Altos V
DEC VT100
DEC VT200
Wyse 50
Wyse 60
Wyse 150
Wyse 160
Wyse 370

Terminals that are not supported or well suited for use with the UNIX operating system functions and utilities include:

Wang 2x36 DE/DW
Wang 2110A

6.3.5 Common Default Keyboard Equivalences

The keys referenced in this section refer to a set of commonly used keys within NPL. Depending on the type of terminal being used, the keyboard may have differently defined keys. The table below provides the default equivalences of these keys for the terminals supported in this release. Refer to Appendix D of the NPL Programmer's Guide for a complete description of default keyboard equivalences for all terminals listed below.

Supported Terminal	**Common Keys**		
	EXECUTE	**CANCEL**	**HELP**
ALTOS III	LINE FEED	HOME	HELP
ALTOS V	LINE FEED	HOME	HELP
DEC VT100	LINEFEED	ESC c	ESC h
DEC VT200	SELECT	FIND	HELP
IBM CONSOLE MONITOR	HOME	END	ESC,ESC
IBM 3151	HOME	BACK TAB	ESC,ESC
WANG 2110A	EXECUTE	CANCEL	HELP
WANG 2x36 DE/DW	RUN	CANCEL	SHIFT,RESET
WYSE 50 (ASCII Style KB)	SEND	DEL	REPLACE
WYSE 60 (ASCII Style KB)	SEND	DEL	REPLACE
WYSE 60 (PC Style KB)	HOME	END	ESC,ESC
WYSE 150 (ASCII KB)	SEND	DEL	REPLACE
WYSE 150 (PC Style KB)	HOME	END	ESC,ESC
WYSE 160 (ASCII Style KB)	SEND	DEL	REPLACE
WYSE 160 (PC Style KB)	HOME	END	ESC,ESC
WYSE 370 (ASCII Style KB)	SEND	DEL	HELP

6.4 Monochrome Console Characteristics

The following section discusses the console terminal characteristics using a monochrome controller.

6.4.1 Graphics Availability

The -G (graphics) RunTime startup option is not supported under UNIX.

6.4.2 Screen Character Set

Downloadable Fonts

The console terminal does not support the use of downloadable fonts.

Alternate Character Set (Pixel Graphics)

Use of the alternate character set is not supported by the console terminal.

Non-English Character Sets

Use of non-English character sets through the $OPTIONS Font Designator is not supported on the console terminal.

Screen Translation

The default values for both screen and keyboard translation for the console terminal are not built into the RunTime program. The file SCREEN.imon (and KEYBOARD.imon) must be present, and these will be used by the RunTime instead of built in defaults. Screen translation default values are the same as for an IBM PC.

6.4.3 Box Graphics Support

The console monitor does not support "true" box graphics under NPL. "Character" box graphics are supported. To use "character" box graphics it is necessary to modify byte 1 of the NPL $BOXTABLE system variable. The default value of byte 1 is HEX(00) and indicates the "character" boxes are disabled. A value of HEX(01) enables "character" boxes.

Refer to the NPL Statements Guide for details on the use of the $BOXTABLE system variable.

6.4.4 Attributes Support

Under UNIX, NPL screen attributes on the console monitor operate as documented in Chapter 7 of the NPL Programmer's Guide with the following exceptions:

- Inverse video does not work in conjunction with other video modes. When inverse video is combined with another video attribute, only the inverse video attribute appears.

6.4.5 Cursor Appearance

The console monitor does not have the ability to display a "steady" cursor. A steady cursor will appear the same as a blinking cursor.

6.4.6 Support for 132-Column Mode

The console terminal does not support 132-column mode.

6.4.7 Use With Native Operating System Functions and Utilities

The console terminal is supported (and well suited) for use with UNIX native operating system functions and utilities.

6.5 Color Console Terminal Characteristics

The following sections discuss the console terminal characteristics for the Color Graphics Controller.

6.5.1 Graphics Availability

The -G (graphics) RunTime startup option is not supported under UNIX.

6.5.2 Screen Character Set

Downloadable Fonts

The console terminal does not support the use of downloadable fonts.

Alternate Character Set (Pixel Graphics)

Use of the alternate character set is not supported by the console terminal.

Non-English Character Sets

Use of non-English character sets through the $OPTIONS Font Designator is not supported on the console terminal.

Screen Translation

The default values for both screen and keyboard translation for the console terminal are not built into the RunTime program. The file SCREEN.imon (or KEYBOARD.imon) must be present, and these will be used by the RunTime instead of built in defaults. Screen translation default values are the same as for an IBM PC.

6.5.3 Box Graphics Support

The console monitor does not support "true" box graphics under NPL. "Character" box graphics are supported. To use "character" box graphics it is necessary to modify byte 1 of the NPL $BOXTABLE system variable. The default value of byte 1 is HEX(00) and indicates the "character" boxes are disabled. A value of HEX(01) enables "character" boxes.

Refer to the NPL Statements Guide for details on the use of the $BOXTABLE system variable.

6.5.4 Attributes Support

Under UNIX, NPL screen attributes on the console monitor operate as documented in Chapter 7 of the NPL Programmer's Guide with the following exceptions:

- Inverse video does not work in conjunction with other video modes. When inverse video is combined with another video attribute, only the inverse video attribute will appear.

- Underlines are not supported on the IBM color graphics monitor. Under SCO and AT&T UNIX, the control sequence for "underline" is ignored on color consoles. However, on Interactive UNIX, the "underline" control sequence is recognized on color consoles and generates a bright white foreground on red background combination. This color combination cannot be changed by the application.

6.5.5 Cursor Appearance

The console monitor does not have the ability to display a "steady" cursor. A steady cursor will appear the same as a blinking cursor.

6.5.6 Support for 132-Column Mode

The console terminal does not support 132-column mode.

6.5.7 Use With Native Operating System Functions and Utilities

The console terminal is supported (and well suited) for use with UNIX native operating system functions and utilities.

6.6 Color Support under NPL

When color is used by the application, NPL sends ANSI standard color control sequences to the console. Some versions of Intel UNIX do not provide ANSI compatible color support for the monitor. On these systems, color support will be erratic. Incorrect colors may be displayed.

NOTE: Color support is provided on the Wyse 370 color terminal. Refer to Appendix D of the NPL Programmer's Guide for more information.

6.7 Graphics Support under NPL

The -G (graphics mode) option is not supported under the UNIX implementation of NPL. True box graphics are only supported on the Wang 2236 terminal. Refer to Appendix D for terminal specific information and Section 7.4.7 for NPL color support of the NPL Programmer's Guide for details on

6.8 Keyboard Characteristics

The IBM AT console should be configured to use UNIX keyboard table "KEYBOARD.imon" and not MS-DOS keyboard tables.

Default equivalences for commonly used keys are:

HALT	CTRL/C (configurable by the UNIX stty intr command)
EXECUTE	HOME
CANCEL	END
HELP	ESC,ESC

The following editing keys are available:

Keypad arrows	(NORTH, SOUTH, EAST, and WEST)
Right arrow	Recalls the previous command entry. If a line number is entered before pressing the right arrow key, that specific program line will be recalled.
?	Inserts a linefeed within a line of text.
?	Deletes characters from the current cursor position to the end of line.
INSERT	Inserts a blank space within a line of text.
DELETE	Deletes a character at the current cursor position.
PREV/PAGE	Causes the screen display to shift to the previous screen (if more than 23 lines of text exist) or cause cursor movement to the first line of text.
NEXT/PAGE	Causes the screen display to shift to the next screen (if more than 23 lines of text exist) or cause cursor movement to the last line of text.
CTRL-P	Recalls the previous command from the "multi-command" keyboard buffer. Refer to Chapter 5 of the Programmer’s Guide for more information.
CTRL-N	Recalls the next command from the "multi-command" keyboard buffer. Refer to Chapter 5 of the NPL Programmer’s Guide for more information.

In Edit Mode, SF keys are assigned the following functions:

F5	Moves the cursor to end of line being edited.
F6	Moves the cursor down one line.
F7	Moves the cursor up one line.
F8	Moves the cursor to the beginning of line being edited.
F9	Erases all text from current position to end of line.
F10	Deletes one character at the current cursor position (use DELETE under Interactive and AT&T UNIX).
SHIFT/CTRL F1	Inserts one space at the current cursor position. (use INSERT under Interactive and AT&T UNIX).
SHIFT/CTRL F2	Moves the cursor 5 spaces to the right. (use F12 under Interactive and AT&T UNIX).
SHIFT/CTRL F3	Move the cursor one space to the right. (use EAST under Interactive and AT&T UNIX).
SHIFT/CTRL F4	Moves the cursor one space to the left. (use WEST under Interactive and AT&T UNIX).
SHIFT/CTRL F5	Moves the cursor 5 spaces to the left. (use ESC,r under Interactive and AT&T UNIX).
SHIFT/CTRL F6	Recalls the current line. (use SHIFT/F4 under Interactive and AT&T UNIX).

6.8.1 Default Equivalence Table

When determining a key sequence in the default Keyboard Equivalences Table, the following terminology is used. When a "-" (dash) is used in a key sequence, it indicates that the keys should be pressed simultaneously. When a "," (comma) is used in a key sequence, it indicates that the keys should be pressed in sequential order. A "/" is used to separate different key sequences that are assigned to one NPL virtual key. For example, CTRL-x indicates press and hold the CTRL key while pressing x; ESC, x indicates press and release the ESC key, then press x, and; SHIFT-PREV / CTRL-P indicates that both the SHIFT-PREV and the CTRL-P perform the same function.

UNIX Console Compatible Default Keyboard Equivalences Tale

NPL Code Key	NPL Virtual Key	Console Terminal Key
08	BACKSPACE	BACKSPACE
0D	RETURN	ENTER
5F	?	UNDERLINE
81	CLEAR	?
82	EXEC	7-HOME
84	CONTINUE(LOAD)	?
A1	LOAD	CTRL-X
E5	SHIFT-ERASE	CTRL-W
FF'A0'xx	UNDERLINE(DEAD KEY)	?
'00 ... '09	SF '0 ... '9	F1 ... F10 UNIX Function keys (F1-F10) Use F1...F12 under Interactive and AT&T UNIX.
'0A ... '0F	SF '10 ...'15	SHIFT/CTRL F1 ... F6 UNIX Function keys (F37-F42) Use ESC, (w,e,r,t) under Interactive and AT&T UNIX.
'10 ... '19	SHIFT SF '0...'9	SHIFT F1 ... F10 UNIX Function keys (F13-F22) Use SHIFT F1...F12 under Interactive and AT&T UNIX.
'1A ... '1F	SHIFT SF'10...'15	CTRL F1 ...F6 UNIX Function keys (F25-F30) Use ESC, SHIFT (W,E,R,T) under Interactive and AT&T UNIX.
'42	PREV-SCREEN	9-PG UP
'43	NEXT-SCREEN	3-PG DN
'45	SOUTH	2-SOUTH
'46	NORTH	8-NORTH
'48	ERASE	CTRL-E
'49	DELETE	./DEL
'4A	INSERT	0/INS
'4C	EAST	6/EAST
'4D	WEST	4/WEST

UNIX Console Compatible Default Keyboard Equivalences Tale		
NPL Code Key	**NPL Virtual Key**	**Console Terminal Key**
’4F	RECALL	CTRL-R
’50	SHIFT-CANCEL	CTRL-K
’52	SHIFT-PREV-SCREEN	CTRL-P
’53	SHIFT-NEXT-SCREEN	CTRL-N
’55	SHIFT-SOUTH	?
’56	SHIFT-NORTH ESC, NORTH	?
’59	SHIFT-DELETE (LINE DEL)	?
’5A	SHIFT-INSERT (LINE INS)	?
’5C	SHIFT-EAST (EAST-5)	?
’5D	SHIFT-WEST (WEST-5)	?
’5F	D TAB	CTRL-T
’7C	GL	CTRL-G
’7D	SHIFT-GL	CTRL-Z
’7E	TAB	TAB
’7F	SHIFT-TAB	SHIFT-TAB
’E1	HELP	ESC, ESC
’F0	EDIT	1-END

A question mark (?) indicates that no key is assigned to the NPL code. The column "Console Key" represents default values under UNIX.

when a "-" (dash) is used in a key sequence it indicates that the keys should be pressed simultaneously. when a "," (comma) is used in a key sequence it indicates that the keys should be pressed in sequential order.

CHAPTER 7

MULTI-USER CAPABILITIES

7.1 Overview

Multi-user capabilities are supported for NPL under UNIX. This chapter provides a summary of these capabilities.

Section 7.2 discusses device sharing and "hogging"

Section 7.3 discusses global partitions.

Section 7.4 discusses terminal identification.

Section 7.5 discusses intertask communications.

Section 7.6 discusses the user count.

Section 7.7 discusses network operation.

Section 7.8 discusses multi-screen products.

7.2 Device Sharing and "Hogging"

Device sharing and "hogging" is commonly performed by use of the NPL statements, $OPEN and $CLOSE. In some cases, $GIO statements may also be used. The $OPEN and $CLOSE statements, when directed to disk devices, are fully implemented and functional in NPL for UNIX. This is also true when directed to PRINT class devices (refer to Section 5.5 for the discussion on printer handling). $GIO statements which perform device "hogging" are not supported.

NOTE: Due to differences between NPL and the Wang 2200's handling of $OPEN when multiple devices are specified and one of the devices is "hogged" by another user, a $OPTIONS byte has been implemented to apply either Wang logic or Niakwa logic to the $OPEN operation. Refer to Section 8.2.2 of this Supplement for more information on this option.

Refer to the NPL Statements Guide for more information on the exact syntax and use of the $OPEN and $CLOSE statements.

7.2.1 Suppress Implicit $OPEN Logic

Byte 39 of $OPTIONS can be used to specify that no implicit $OPEN is to be performed on DATA LOAD BA and DATA LOAD BM when the platter has not been explicitly hogged by $OPEN.

Byte 39 of $OPTIONS may contain the following values:

HEX(00) (default)	Indicates the implicit $OPENs are to occur as they have in prior releases.
HEX(01)	Indicates that implicit $OPENs are to be suppressed.

HINT: Niakwa highly recommends use of this feature. It should increase performance for applications that do not use explicit $OPENs. In particular, performance will be improved in environments such as NFS where file lock calls have a heavy performance penalty.

7.2.2 Suppress $OPEN on Read Only Files

NPL automatically suppresses all file lock calls to files that are Read Only. Under UNIX, Read Only is defined as no user having write ("w") privileges. File privileges can be examined using the ls -l command. File privileges can be modified using the chmod command. For example, to remove write privileges from a file, enter the following from a UNIX prompt:

```
chmod a-w filename
```

For files that can be Read Only, such as program diskimages, this will improve performance, particularly on systems such as NFS where file lock calls have a heavy performance penalty.

7.3 Global Partitions

Global partitions are not supported by NPL. However, global variables are supported. Applications that depend upon the global partitions for record locking must be modified to use a disk-based record-locking system.

HINT: Placing lock files in a RAM disk will substantially improve performance. Refer to Section 5.3.4 for details.

NOTE: **Applications which can handle a multiplexed 2200 environment will likely handle record locking without the global partition since in a multiplexed 2200 environment, the global partition cannot be used for record locking.**

7.4 Terminal Identification

Under UNIX, NPL has the ability to uniquely identify individual terminals through the use of system variables #TERM, #PART, and #ID. This allows multi-user applications to share information, while maintaining data integrity, by having control over individual terminals.

7.4.1 #TERM

Terminal identification (#TERM) is determined by finding the terminal device name in the "ttys" file in the current directory. If the ttys is not found, the RunTime checks for this file in the directory specified by the NIAKWA_RUNTIME environment variable and finally in the /usr/BASIC2C directory. If this file is not found at these locations, the RunTime tries to use the system default file /etc/ttys.

NOTE: On some UNIX operating systems, the /etc/ttys file does not exist. On these systems, the /usr/BASIC2C/ttys file must be set up. The NPL RunTime program will not execute if it cannot determine the terminal number.

An example ttys file (/usr/BASIC2C/ttys.exp) is provided on the Terminal Support Files Diskette contained in the NPL Development Package.

Example ttys Use

The following is an example of how this file is used.

Assume the system is configured as follows:

```
16console        #TERM=1
06tty01          #TERM=2
T16tty02         #TERM=3
16tty03          #TERM=4
16tty04          #TERM=5
16tty05          #TERM=6
06lpT            #TERM=7
```

Assuming tty01 and lp are both printers (both unusable as NPL terminals) and that tty03 is restricted from accessing NPL, the /usr/BASIC2C/ttys file would be set up as follows:

```
16console        #TERM=1
16tty02          #TERM=2
T16tty04         #TERM=3
16tty05          #TERM=4
```

NOTE: The first blank or tab is treated as the start of a comment and #TERM=x comments can be included in the ttys.

The terminal number can also be set explicitly by using the -T command line option to rti/rtp. Refer to Section 2.4.11 in the NPL Programmer's Guide for details and Section 8.3.1 and 8.3.2 in this Supplement for the use of the -T option for a background partition.

If the -T option is not set, and the device name is not found in the appropriate ttys file, the NPL RunTime exits with the message "cannot determine terminal number".

7.4.2 #PART

#PART is maintained for compatibility with other systems on which it is used to indicate the physical partition of memory that the RunTime process is running in.

Under UNIX, #PART for a foreground partition has the same value as #TERM. For background partitions, #PART has the value specified by the -B start up option. Refer to Section 8.3 for further details on background partitions.

7.4.3 #ID

NPL supports a system variable (BASIC2C_ID) to establish #ID values for the RunTime. This provides a method of unique terminal identification on UNIX networks.

NOTE: Niakwa currently does not officially support the use of UNIX network environments. NPL is expected to work well in a network environment, but Niakwa has not evaluated any UNIX networks and has no plans to do so.

The BASIC2C_ID variable can be set by entering BASIC2C_ID=x (where x is a numeric value between 1 and 255) at the UNIX command line or in a shell script. The following is an example of how to set the BASIC2C_ID system variable in a user .profile file:

```
#Sets the BASIC2C_TERM for a Wyse 50 Terminal.
    BASIC2C_TERM=wy50
#Sets the BASIC2C_ID for this terminal.
    BASIC2C_ID=5
#Exports both BASIC2C_TERM and BASIC2C_ID
    export BASIC2C_TERM BASIC2C_ID
#Sets the UNIX Interrupt key to Control/C
    stty intr '^C'
```

The above lines that begin with a # are comment lines and are ignored by UNIX.

7.5 Inter-task Communication

The following section discusses inter-task communication options for NPL under UNIX.

7.5.1 $PSTAT

NPL applications can use $PSTAT to pass information between partitions.

The $PSTAT statement is a special instruction that returns various status information for the partition specified in the expression. The argument for the function must be specified as a numeric expression equal to the partition number.

An alpha variable can be set to the contents of $PSTAT. This variable contains the following:

Byte	Contents
1-8	User-defined area
9	Operating system type
10	RunTime revision number
11	Bank # - set to #PART in packed (##) format
12-13	Always contains SPACEK in packed (##.##) format
14	Programmability ("P" or " ")
15	Terminal number in packed (##) format
16	Partition status
17-24	Blank
25	ERR function value (numeric portion of the last error encountered (in hex))
26-28	Partition # assignments, respectively
29	Device-address

For example:

```
0010 Q$ = $PSTAT(#PART)
0010 B$(),STR(A$,1,30) = $PSTAT(#PART)
```

Refer to the NPL Statements Guide, $PSTAT, for details on the exact syntax and use of this statement. Refer to Section 8.2 for details on use of $PSTAT with background partitions.

7.5.2 $BREAK

The $BREAK statement is used to relinquish CPU time to another task.

The delay caused by $BREAK is about 20 milliseconds times the number of "timeslices" specified (the default = 1). Under the UNIX implementation of of NPL, all timeslices specified are always released. The amount of time released is rounded to the nearest whole second. This is a variance from the Wang 2200 implementation where each released timeslice may cause a delay varying from 0 (no other activity) to 20 milliseconds times n (where n = the number of other partitions executing and using a full timeslice).

For this reason, $BREAK with large values should be avoided. For example, $BREAK 255 delays for 255 x 20 = 5100 milliseconds, or over 5 seconds.

Byte 14 of $OPTIONS may be used to cause an implicit $BREAK to be performed after disk I/O. Refer to Section 8.2.2 for details.

Refer to the NPL Statements Guide for the exact syntax and use of the $BREAK statement.

NOTE: The special case $BREAK! causes an exit from the RunTime Program.

UNIX Versus Xenix Timeslicing

On UNIX/386 systems, the timeslice algorithm differs from XENIX systems in that the timeslice is much longer than on XENIX systems. This means that $BREAK cannot be used to release small units of time as intended. The effects of this are sometimes subtle since $BREAK is implicitly used by several NPL statements. In particular $BREAK is used implicitly by:

• KEYIN	Polling form.
• $DEMO	
• $CLOSE	Implicit $BREAK set by byte 14 of $OPTIONS.

Since releasing the full UNIX timeslice for each $BREAK unit specified would significantly slow down many applications, the UNIX implementation of NPL, by default, issues one UNIX timeslice (one second) for each 50 $BREAK units, rounded to the nearest integer (e.g., 25=1 second, 74=1 second, 75=two seconds, 100= 2 seconds, 125=3 seconds, 150=3 seconds, 175=4 seconds, and 24 or less is rounded down to 0). Thus, in cases where $BREAK is specified with less than 25 units, no time is released.

For applications that repeatedly loop through a $BREAK (either explicitly or implicitly using one of the statements discussed above) with less than 25 units, this does not provide a good solution. Therefore, byte 36 of $OPTIONS may be used to indicate that the number of $BREAK units released is to be accumulated and that a UNIX timeslice release should occur when the number of units accumulated is equal to or greater than the binary value specified in byte 36 of $OPTIONS. The amount of UNIX timeslices released is equal to the amount accumulated divided by 50. For example, if byte 36 is set to 50, and a $BREAK 40 is done, no timeslices are released. If a $BREAK 165 is then done, it releases 4 seconds (i.e., ((165+40)/50) and any remainder is thrown away (i.e., the accumulated total is reset to 0).

The following describes the possible values of byte 36 of $OPTIONS:

HEX(00)	Default. No accumulation of $BREAK units will occur.
HEX(xx)	Where xx is a value from HEX(00) to HEX(FF). $BREAK will execute a timeslice release when the number of units accumulated is equal to or greater than the binary value of xx.

For example, a value of 25 in byte 36 would indicate that the RunTime would accumulate $BREAK units and would perform a UNIX timeslice release after each 25 accumulated $BREAK units. A value of 100 in byte 36 would indicate that the RunTime would accumulate $BREAK units and would release two UNIX timeslices after each 100 accumulated $BREAK units.

HINT: This solution is not ideal, but it probably is the best that can be done given the inherent limitation of UNIX. For applications where performance problems are noted, some trial and error may be needed to find the best value to use in byte 36 of $OPTIONS. A value of HEX(50) would be a good first try. Applications that rely on a polling KEYIN to input data may require a higher value to provide smooth operator keyboard response, but this will result in using more system resources when the operator is not active, thus slowing down other tasks. If performance problems prove severe, programmers may want to consider use of the non-polling KEYIN instead of the polling KEYIN.

NOTE: This feature is specific to the NPL 386 Runtime operating in a UNIX environment. If used in an environment other than UNIX, byte 36 of $OPTIONS will be ignored.

7.5.3 DATE and TIME

The DATE and TIME functions operate normally with the following exception:

An attempt to execute a DATE= or TIME= statement by any user other than the superuser results in an error X78. This is the same error that is generated if an incorrect password is used in specifying DATE= or TIME= on the Wang 2200.

NOTE: This X78 error occurs even if the password is properly specified.

Use of DATE or TIME functions on the right side of a LET statement are not affected.

7.5.4 System Messages

The broadcast message ($MSG) may be assigned and inspected from any terminal. Refer to the NPL Statements Guide, $MSG, for details.

7.6 User Count

Under UNIX, each terminal using NPL counts as one user against the total user limit. There are no charges against the user count for operating background partitions or for reinvoking the RunTime from a $SHELL statement. Refer to Section 8.3 for details on background partitions.

7.6.1 userfix Utility

The userfix utility contained on the UNIX Runtime diskettes, allows the user to reset the user count in situations where the user count is greater than the actual number of NPL users on the system (typically this occurs when the RunTime is aborted improperly or the UNIX "kill -9" command is used to kill a task.

NOTE: The userfix utility is available only for the UNIX (386) RunTime. It is not available with the Xenix (286) RunTime. If the user count is greater than the actual number of NPL users under Xenix, contact Niakwa for information on how to proceed.

To adjust the user count with this utility, follow the steps shown below:

1. Change directories to the directory where the NPL files are located (typically /usr/BASIC2C).

2. Enter the following:

```
./userfix
```

The program version number and copyright notice are then displayed. The user count before and after the operation of the utility are also displayed with an "End of processing" message. Once completed, the user count is reset to reflect the current number of users in the RunTime.

NOTE: Users currently using NPL are not affected by the operation of this program.

7.7 Use on Networks

Although operation of NPL on UNIX networks has not been tested and is not officially supported by Niakwa, the following guidelines may be useful for those developers who wish to pursue this issue.

- The NPL RunTime security and user limit checks are always performed on the local node. Therefore, each node must have its own NPL RunTime installed for the proper number of users on that node.

- $PSTAT and $MSG are always local. That is, each node has its own value for $PSTAT and $MSG.

- It is the responsibility of the application to ensure that unique logical terminal number identification is generated by use of the BASIC2C_ID system variable described above.

- Files can be accessed across the network by proper specifications of the $DEVICE equivalence. Typically, this involves specification of nodename/pathname/filename. The format for nodename/pathname should be identical to the format required to access the file by any UNIX command (ls, for example).

- All auxiliary files, such as the screen and keyboard tables, the ttys file, the ENABLED file, etc., are accessed locally. Therefore, copies of the files must be present on each node where use of NPL is desired.

- File locking ($OPEN/$CLOSE) is accomplished from standard UNIX system calls.

- Use byte 39 of $OPTIONS to specify that no implicit $OPEN is to be performed with DATA LOAD BA and DATA LOAD BM statements when the platter has not been explicitly hogged by $OPEN. This should increase performance significantly in a UNIX networked environment. Refer to Sections 7.2 and 8.2.2 in this Supplement for more information on this feature.

- Suppress all file lock calls to read only files by setting the file to Read Only. Under UNIX, Read Only is defined as no user having write ("w") privileges. For files that can be Read Only, such as program diskimages, this will improve performance significantly in a UNIX networked environment. Refer to Section 7.2 and 8.2.2 or more information on this feature.

7.8 Multi-Screen Products and Issues

SCO System V/386 UNIX Multi-screen capability gives the user the ability to operate the video display and keyboard as ten separate tasks (tty01 .. tty10). NPL was tested running under these separate tasks. The #TERM value within each task is individually unique from one task to the next.

NOTE: When SCO is configured to run with Multiscreen capability, it causes a conflict with NPL. SCO System V/386 assigns ALT F1-F10 as the means to switch from one task to another. When using a keyboard with twelve function keys, function keys F11 and F12 return the same value as function keys ALT F1 and ALT F2, causing the console to switch tasks. Therefore, these keys cannot be used by NPL when Multiscreen capability is enabled.

CHAPTER 8

PLATFORM-SPECIFIC LANGUAGE FEATURES

8.1 Overview

There are a series of NPL statements which are specific to the type of hardware/operating environment in which they are executed. This chapter discusses the language features specific to NPL under UNIX.

Section 8.2 discusses the environment-specific statements

Section 8.3 discusses background partition support.

Section 8.4 discusses memory management.

8.2 UNIX-Specific Statements

This section discusses the NPL language features that are specific to the UNIX operating system

8.2.1 $MACHINE

The $MACHINE system variable contains information about the hardware environment in which the RunTime is executing. In the UNIX implementation, the following platform-specific values are set:

Byte	Description
Byte 1	RunTime Version: A = UNIX (386) version X = Xenix (286) version.

Byte 2	Hardware Manufacture Code:
	HEX(00) (for IBM)
	HEX(02) (for Altos)

Byte 3	Monitor Type:
	Blank = ASCII terminal
	M = Monochrome
	C = Color (dependent on type of console monitor)

Byte 4	Graphics Enabled - "G" = graphics enabled (if using a Wang 2236 terminal) " " (blank) = "true" box graphics are not available
Byte 5	Hardware Model Code. The model code varies from one manufacturer to another.
Byte 6	Number of users active at boot time.
Byte 8	Display width in binary (80 or 132 column).

Byte	Description
Byte 9	Current Terminal Type: HEX(00) = Wang PC HEX(01) = Wang 2110A HEX(02) = Altos III HEX(03) = VT100/VT200 HEX(04) = IBM PC HEX(05) = Wyse 60,150,160 HEX(06) = Wyse 50 HEX(07) = Wang 2236 HEX(08) = Altos V HEX(09) = IBM AT HEX(0B) = Wyse 370
Byte 10	Math co-processor present. A HEX(00) indicates that no co-processor is present. A HEX(01) indicates that a co-processor is present and may be used for some math operations by setting byte 16 of $OPTIONS to a value of HEX(01).
Byte 12	Number of colors available. Refer to Chapter 6 for details on color support for NPL under UNIX.
Byte 13	Maximum number of authorized users, displayed in binary.
Byte 18	Foreground or Background Task HEX(00) = Foreground HEX(01) = Background
Byte 21	Contains the maximum number of entries (in binary) allocated to the handle table (in K) during a RunTime session.
Byte 25, 26	Extended field equivalent to $MACHINE byte 6 (number of active users in the RunTime before this task booted). For systems with 256 or more users, these bytes must be used to get an accurate user count.
Byte 27, 28	Extended field equivalent to $MACHINE byte 13 (maximum number of authorized users). Fore systems with 256 or more users, these bytes must be used to determine the maximum user count.

NOTE: $MACHINE is a 64 byte variable. The above bytes are specific to the UNIX operating environment. for a complete description of all bytes within the $MACHINE system variable, refer to the NPL Statements Guide, $MACHINE.

8.2.2 $OPTIONS

The NPL RunTime allows for the inspection or modification of the $OPTIONS system variable, which consists of a variety of options. Bytes which have specific meanings under the UNIX version of NPL include:

Byte	**Description**
Byte 7	Keyboard translation complex key lead value. When this value is received by the keyboard handlers from the native operating system, a complex key sequence is assumed to follow. HEX(20 = Blank
Byte 8	Defines terminal functionality type
Byte 9	Trailing code to finish the complex key sequence. HEX(00) = No trailing code
Byte 10	Alternate keyboard translation complex key lead value. When this alternate value is received by the keyboard handlers from the native operating system, a complex key sequence is assumed to follow. HEX(00) = No alternate code follows
Byte 11	Trailing code to finish the complex key sequence. HEX(00) = No trailing code
Byte 15	Font designator for non-English character sets. Refer to the appropriate sections in Appendix D of the NPL Programmer's Guide for specific details on each terminal type.
Byte 16	Switch for using a math co-processor. A value of HEX(00), the default, indicates that the math co-processor should not be used. A value of HEX(01) indicates that the math co-processor should be used.
Byte 21	Controls display of "bright" attribute on terminals where "normal" is actually "dim". Refer to Chapter 6 for information on specific controller/monitors where this feature is supported. A value of HEX(00) indicates that "bright" is displayed as "bright" and "normal" is displayed as "dim". A value of HEX(01) indicates that "bright" is displayed as "dim" and "normal" is displayed as "bright".
Byte 22	Provides power-on default background/foreground color selection for supported color terminals. Refer to Chapter 6 for details. HEX(00) is the default - no color.
Byte 23	Provides power-on default perimeter/underline color selection for supported color terminals. Refer to Chapter 6 for details. HEX(00) is the default - no color.

Byte	**Description**
Byte 31	Controls certain features for terminal emulators which do not provide 100% support of the terminal being emulated. This feature is provided strictly as a convenience for the users who must use emulation products. These emulation products are not supported for use with NPL. Each defined bit can be used to suppress an NPL feature for the terminal in use. Defined bits include: HEX(00) Default - No features supplied. HEX(01) The HELP display highlights only the current field. HEX(02) The terminal has no local printer capability. HEX(04) 132 column mode is not supported.
Byte 36	Indicates that the number of $BREAK units released is to be accumulated and that a UNIX timeslice release should occur when the number of units accumulated is equal to or greater than the binary value specified in this byte. HEX(00) Default - no UNIX timeslice is released
Byte 37	Specifies if the characters input as response to LINPUT and INPUT statements are echoed by NPL. If set to HEX(01) and the UNIX echo parameter is set OFF before entering the RunTime, no characters are echoed to the screen when keys are pressed in response to the LINPUT or INPUT statements. IF echo is on before entering the RunTime, then the standard LINPUT or INPUT is used despite the value of this byte. When this byte is set to any value other than HEX(01), the standard LINPUT or INPUT is used despite the status of the UNIX echo parameter. No other NPL statements are modified. The operation of KEYIN is not affected by this byte. HEX(00) - No suppression takes place. HEX(00) Default - Standard input. HEX(01) No characters echoed. By using the status command on the echo parameter, the developer is able to avoid complex logic in determining whether to set byte 37 of $OPTIONS on sites where some terminals are local and should echo characters normally while others are operating remotely and the echoing should be suppressed. The normal setting for the echo parameter is ON in UNIX environments.

Byte	**Description**
Byte 39	Used to specify that no implicit $OPEN is to be performed on DATA LOAD BA and DATA LOAD BM when the platter has not been explicitly hogged by any $OPEN. HEX(00) Default. Implicit $OPENs are to occur as they have in prior releases. HEX(01) Implicit $OPENs are suppressed.
Byte 46	Allows customization of the HELP processor's use of keys: HEX(01) - 0 Return = TAB/EAST HEX(01) - 1 Return = EXEC

NOTE: $OPTIONS is a 64 byte variable and must be treated as such or unpredictable results may occur. Refer to the NPL Statements Guide, $OPTIONS, for details on the exact syntax and use of this statement, as well as the contents of the remaining bytes of the variable.

8.2.3 $PSTAT

Refer to the NPL Statements Guide, $PSTAT, for details on the exact syntax and use of this statement.

8.2.4 $SHELL

All NPL tasks have $SHELL (INVOKE) capabilities. This allows a temporary exit from the NPL environment to allow for interfacing with UNIX functions and commands.

When a $SHELL operation is executed, a new UNIX shell process is started and control is transferred to it. The shell used is always the standard shell (/bin/sh) despite the value of the SHELL system variable.

Because the invoke operation runs as a child process to the process executing the NPL RunTime, any changes made to the process' environment, such as changing the current directory or the value of system variables, do not affect the environment of the NPL RunTime itself. For example, if the NPL RunTime was invoked from the /usr/BASIC2C directory, and the following commands were entered:

```
!cd /usr/ARPROGS; pwd
!pwd
```

the first "pwd" (print working directory) command would print "/usr/ARPROGS", and the second one would print "/usr/BASIC2C". This occurs because each process under UNIX has a unique "environment" (set of system variables), which is copied from its parent process when the new process is started. If the new process changes its system variables, those of the parent process are not affected.

NOTE: The effect of changing process information from within $SHELL may vary between different NPL-supported platforms. For example, under MS-DOS, changing the current directory with $SHELL changes the current directory within NPL. Programmers developing NPL applications for multiple platforms should take this into account when using $SHELL.

8.3 Background Partition Support

The background partition capability of NPL is supported under UNIX. This makes it possible to run multiple occurrences of the NPL RunTime program as separate UNIX processes, all attached to the same terminal. This section discusses the implementation and use of NPL background partitions under UNIX.

8.3.1 Configuration Requirements

The following details the use of background partitions under UNIX.

- The NPL RunTime may be directed to run as a background task by specifying the -B option on the rti (or rtp) command line. The background task may be started in the user's .profile file, from any UNIX command line, or from within a currently running NPL RunTime by using $SHELL. The -B option informs the RunTime that it is to operate in a background partition and what partition number it is assigned.

The following example starts the Interpretive RunTime program as a background partition with a #PART value of 25 and instructs it to run the boot program /usr/BASIC2C/MYBOOT.OBJ:

```
rti -B=25 /usr/BASIC2C/MYBOOT
```

This example illustrates starting a background RunTime partition from the $SHELL command:

```
$SHELL "rti -B=25 MYBOOT"
```

NOTE: The NPL RunTime does not check for duplicate #PART values. It is the responsibility of the application to ensure the #PART values specified for any background task are unique.

- The -T option may be used to associate the background partition with a specific port. Otherwise, the background partition is associated with the port from which it is started. The #TERM value is the same for all partitions, background and foreground, associated with the same port.

NOTE: Background tasks can be started from either foreground or background and do not have to be associated with a specific port if the -T option is used (refer to Section 7.4.1 for more information).

- Multiple background partitions may be assigned to the same port.

8.3.2 Operation of the Background Partition

The following should be considered when using the background partition.

- Write attempts to the screen by the application when running in a background partition are stored in an internal buffer. When the partition is switched to foreground, the buffered screen contents are displayed.

- Values for byte 16 of $PSTAT (terminal status byte) are maintained as follows:

  - "A" for partitions in foreground.

  - "D" for partitions in background, which are not waiting for a terminal to be attached.

- "W" for background partitions, which are waiting for a terminal to be attached. This state is caused by execution of a non-polling keyboard input request by the background partition.

- #PART is equal to the value assigned with the -B option. Refer above for details.
- #TERM is generated based on the /usr/BASIC2C/ttys file. #TERM is identical for all partitions initiated from a given terminal, whether foreground or background. The -T option may be used to assign a specific #TERM value. Refer to Section 7.4.1 for more information.
- Byte 15 of $PSTAT (terminal number) is handled like #TERM described above.
- $RELEASE TERMINAL is supported with restrictions as noted in Section 8.3.3 below.
- $IF ON directed to addresses 005 or 001 returns a status of "BUSY" if a terminal is not attached to the partition or a status of "READY" if a terminal is attached to the partition. This is consistent with operation on the Wang 2200.
- Terminal type is determined by the terminal type of the foreground partition.

8.3.3 Restrictions

The following restriction apply the background tasks:

- Background partitions cannot be assigned to terminal number zero. Each background partition must be associated with a specific terminal and may be accessed only from that terminal. If the -t option is used to assign a specific #TERM value, then the background partition can only be accessed from the terminal specified.
- $RELEASE PART performs no operation. On the Wang 2200, $RELEASE PART releases the current partition to terminal number zero, so it may be accessed by another terminal. Since in NPL background partitions must be associated with a specific terminal, this statement is meaningless.
- Printing to local printers from a background partition is not supported. Attempting to access the local printer results in a P48-Illegal Device Specification error. Printing to system printers or ASCII text files from the background partition is fully supported.

- Limitations for $RELEASE TERMINAL:

  - $RELEASE TERMINAL TO x is successful only when the background partition specified is waiting for keyboard input.

  - The HALT key may not function properly when a background task is in the foreground.

8.4 Memory Management

This section discusses Release IV memory allocation as it pertains to the UNIX implementation of NPL. For additional information concerning RunTime memory usage, refer to Chapter 3 of the NPL Programmer's Guide and Chapter 2 and Section 4.5 of this Supplement.

8.4.1 UNIX Considerations

Most UNIX environments use paging for memory management. Paging involves the copying of small portions of memory (pages) in and out of memory as they are needed by the application. The maximum task size is determined by parameters specified during the installation of the operating system. This value is usually much greater than the amount of physical memory available to the system.

Xenix operating systems, designed to run on an 80286 processor, use swapping instead of paging for memory management. This involves copying the user task back and forth from memory to disk whenever physical memory is exceeded. When swapping is used, the maximum task size is limited to the amount of physical memory available to the system.

8.4.2 Dynamic Partition Size

All NPL program code and defined variables reside within a section of memory defined as the "user partition." The maximum size of the user partition is limited by the task size of the host operating system (see previous section).

Due to the dynamic nature of memory allocation by UNIX, the NPL RunTime allocates an initial 141K to the user partition. This is the minimum size of the user partition and is returned by the SPACEW function. The minimum allocation is used because attempting to allocate the full amount of virtual memory available would cause performance degradation.

At any given time, the amount of memory currently available within the user partition is returned by the SPACEF function. When the value of SPACEF drops below 64K, NPL automatically attempts to allocate another 64K of memory to the user partition. The result of this increase is reflected by a 64K increase in SPACEW. If the RunTime is unable to allocate the memory, the value of SPACEF then drops below 64K.

To illustrate this, consider the following:

UNIX RunTime Environment	**SPACEW Value**	**SPACEF Value**
Initial RunTime Memory	139664	139456
Allocate 64K Variable	139664	73920
Allocate 8300 Bytes	139664	65600
Allocate 2nd 64K Variable	205168	65600*
Allocate 20K Variable	205168	45080**

* Additional 64K segment allocated to user partition.

** Value of SPACEF below 64K indicates the NPL RunTime was unable to allocate the extra 64K.

NOTE: **Once memory is allocated in this fashion, it is never released. Thus, in the example above, executing a CLEAR statement results in both SPACEW and SPACEF reporting 205168. The reason for this is very simple: UNIX does not provide a method of deallocating memory.**

CHAPTER 9

COMPILER OPERATION

9.1 Overview

This chapter provides an overview of the general operation of the NPL Compiler (b2c) under Intel UNIX. For a complete discussion of the NPL Compiler, refer to Chapter 14 of the NPL Programmer's Guide.

Section 9.2 discusses how to invoke the NPL Compiler.

Section 9.3 discusses file-naming conventions under Intel UNIX based operating systems.

Section 9.4 discusses the print devices available under the NPL Compiler.

Section 9.5 discusses the use of UNIX script files to operate the NPL Compiler.

9.2 Invoking the Compiler

The NPL Compiler may be invoked in one of two ways. The first and most common form of invoking the compiler is using the command line method.

For example:

```
    b2c -srcloc /usr/BASIC2C/2200DISK.BS2 PROG1 PROG2
```

the designation of "b2c" at beginning of the UNIX command line, refers to the name of the NPL compiler and causes UNIX to invoke it. Upon execution, the compiler inspects the balance of the command line to determine the desired compiler options and the list of input programs to compile. All compiler options are discussed fully in Chapter 14 of the NPL Programmer's Guide.

The second method of invoking the compiler is to enter "b2c" at the UNIX command line with no specified programs to compile. This implicitly invokes the use of the user-friendly compiler display. This display provides an easy-to-use, alternate method of specifying compiler parameters. Refer to Section 14.19 of the NPL Programmer's Guide for a detailed explanation of the user-friendly compiler display.

9.3 File Naming Conventions

File naming conventions vary from one operating system to the next. This section discusses the Intel UNIX conventions and their implications to NPL.

9.3.1 Supported Characters

The file-naming conventions of UNIX are not entirely compatible with the names of files within NPL diskimages. Specifically, within diskimages, filenames may be any eight characters, with no restrictions. UNIX allows fourteen (14) character filenames, but the permitted character set is restricted. Only the letters A-Z, a-z, 0-9, and a few additional special characters are allowed. Refer to the UNIX documentation for more information.

Consequently, situations may arise where it may not be possible to specify the exact form of an input program name on the command line. For example program names with embedded blanks are legal in NPL diskimages but are not allowed under the UNIX operating system. The problem has no general solution, but it can be circumvented with the TRANSLATE option.

The TRANSLATE Option

The TRANSLATE option is used to inform the compiler of the assumed character equivalences between UNIX filenames and filenames within a diskimage file. The TRANSLATE verb is followed by pairs of characters, each preceded by an "=" character. Each group specifies a from/to pair of hexcodes, the first two hexdigits representing the UNIX character, the second two representing the equivalent character to be used for filenames within diskimage files.

For example:

```
-translate 5F20
```

informs the compiler that underline characters (HEX(5F)) in UNIX filenames are equivalent to space characters (HEX(20)) in names of files within NPL diskimage files. Consequently, whenever the compiler needs to place names of files from within NPL diskimage files directly into an UNIX directory, it replaces the embedded blanks in the filenames from within the diskimage file with underlines. Conversely, when the compiler needs to place an UNIX filename into a diskimage, it replaces underlines with spaces.

HINT: For legibility, groups of codes may be separated by a decimal point (".").

For example:

```
-translate 5F20.7B3C.7D3E.=%=/
```

Defines the following equivalences (where D.I. stands for Diskimages):

UNIX Filenames		**Filenames in Diskimage**	
HEX	**Display**	**HEX**	**Display**
5F	7B	7D	25
_ (underscore)	{	}	%
	<	>	/
20	3C	3E	2F

NOTE: Trailing spaces in a filename are not considered to be part of the name. For obvious reasons, use of filenames which use only upper and lower case alphanumeric characters is preferred in the UNIX environment and is strongly encouraged.

For a complete discussion of the NPL Compiler, refer to Chapter 14 of the NPL Programmer's Guide.

9.3.2 Case Sensitivity

Since the NPL compiler may compile from ASCII text files, the fact that UNIX is case sensitive has several implications when compiling to or from ASCII text files:

1. The compiler only recognizes an uppercase .SRC extension. When creating an ASCII text file using a text editor or copy program, be sure to make the .SRC extension uppercase. The compiler always produces .OBJ and .SRC files in uppercase.

2. Program names must be specified correctly relative to case. If there is a program named BOOT.SRC and it is specified as boot.SRC, the compiler cannot find it.

3. The names of diskimage files must be specified correctly relative to case.

9.3.3 Wildcard Usage

Beyond single program names, the compiler accepts "wildcard" names. The presence of a wildcard name in the input program list causes the compiler to compile all programs in the specified directory or diskimage (as specified with srcloc) that "match" the wildcard pattern. A wildcard name is any program name which contains at least one of the "?" or "*" wildcard characters.

A "?" in a wildcard matches any single character in the same position of the input program name.

A "*" in a wildcard matches any characters or no characters to the end of the program name.

However, since the UNIX shell analyzes the entire command line and replaces all wildcards with filenames before passing control to the compiler, wildcards or program names containing wildcard characters must be entered in single quotes. For example:

'AR?UPD'	Matches "AR1UPD", "ARXUPD", "AR@UPD", but not "AR1UPD2".
'AR*'	Matches all file names beginning with "AR".

```
    b2c -srcloc /dev/fd048ds9 -objloc PLATTER1.BS2 '*'
```

Compiles all programs on a 360K format diskette in drive 0 placing compiled output in diskimage file PLATTER1.BS2 in the current directory.

```
    b2c -srcloc /dev/fd048ds9 -objloc PLATTER1.BS2 'AR??001'
```

Compiles all programs with the first two characters "AR" and the ending characters as "001" from the raw 360K format diskette in drive 0 to the diskimage file PLATTER1.BS2.

NOTE: Do not enter wildcard program names in single quotes when using the compiler display. When using the compiler display, the compiler program (b2c) is in control and the UNIX shell cannot perform any filename substitutions. However, if a shell script file is used which invokes the compiler display, any wildcard program names in the shell script file must still be enclosed in single quotes. The quotes are removed when the display is generated.

Failure to enclose wildcard program names in single quotes at the UNIX command line or shell script files causes all filenames which match the wildcard in the current directory to be passed on to the compiler as program names, including even those files which do not have a .SRC extension.

When wildcard scanning a directory, the scanner automatically adds the .SRC extension. When wildcard scanning a diskimage, the scanner ignores data files even if they match the wildcard pattern.

9.3.4 Pathnames

The NPL compiler (b2c) assumes that all input and output files reside in the current directory. This is modifiable by specifying an alternate pathname in the compiler command line.

For example:

```
    b2c -srcloc /usr/PROGS/PLATTER1.BS2 -objloc /usr/PROGS/PLATTER2.BS2 '*'
```

would locate the "PROGS" directory in the /usr directory and compile all the programs found in PLATTER1.BS2, into PLATTER2.BS2 in the "PROGS" directory.

Output for any of the compiler operations can be discarded by specification of the null device as the output parameter. Under UNIX, the null device must be specified as /dev/null.

For example:

```
b2c -srcloc /usr/PROGS/PLATTER1.BS2 -objloc /dev/null -lstloc /source -
lstformat 2200 '*'
```

Creates files in 2200 atomized format, directly from a diskimage file containing already compiled programs (/usr/PROGS/PLATTER1.BS2). The /dev/null designator for the -objloc option is used to suppress the generation of a second set of compiled programs. Refer to the discussion on the "lstloc" option in Section 14.9 NPL Programmer's Guide.

9.4 Print Devices

Hard copy listings may be produced by specifying -lstloc as the name of a native operating system print-device. The following device designations are valid print device names for the UNIX operating system.

-lstloc lpx	Direct listing to a preconfigured printer device. Where "x" indicates the device number to use. Refer to Section 5.5 for details on configuring printers within the system for use with the NPL.
-lstloc ttyxx	Direct listing to a preconfigured serial device. Where 'xx' indicates the serial port number to use.
-lstloc /dev/null	Suppresses listing.
-lstloc /pathname	Store source listing for each file in the specified directory. Filenames are derived from the input program name, with an extension of .LST or .SRC depending on the -lstformat designation.
-lstloc /pathname/filename	Concatenates all program listings into the specified filename (with a .SRC file extension) in the specified directory.

9.5 Shell Scripts

Shell script files are used to set up and invoke UNIX commands and parameters. Shell script files can be used to invoke the NPL compiler. For example, the following example compiler command line:

```
b2c -srcloc PLATTER1.BS2 -objloc PLATTER2.BS2 -lstloc . -lstformat
.SRC 'AR*'
```

can be set up as a shell script called COMPAR, and then be executed from the command processor by entering the name of the script file:

```
COMPAR
```

Here, the shell script file would perform the same as the example command line it was taken from.

It is possible to generalize shell script files so that some options may be entered during execution of the shell script file. This is handy for specifying the list of input program names to compile. This can be done by a simple modification to the example above. The modification would be to replace the literal program name designation with a replaceable parameter value ("$*") which allows any number of program names to be specified.

NOTE: The quotation marks surrounding the parameter are required.

For example:

```
b2c -srcloc PLATTER1.BS2 -objloc PLATTER2.BS2 -lstloc . -lstformat
.SRC "$*"
```

This batch file would now be able to accept up to any number of program names. For example:

```
COMPAR AR* GL101 OE*
```

Compiles the following sets of programs:

- All programs with the prefix "AR".
- Program GL101.
- All programs with the prefix "OE".

The same compiler options are used for all programs.

9.5.1 Example Shell Script Files

Niakwa has provided several shell script files as examples. These examples are located in the /usr/BASIC2C directory and may be invoked from the UNIX command processor by name. These script files were designed to meet the most frequent requirements of the programmer.

The following are the four script files supplied by Niakwa:

b2ca

```
b2c -display on -srcloc /dev/fd048ds9 -objloc PLATTER1.BS2 -translate
5F20.252F "$*"
```

The above script file compiles input programs from a raw diskette in 360K format, creating p-code files in UNSCRAMBLED format which are placed in the diskimage file PLATTER1.BS2 in the current directory, displaying any compiler warning messages on the screen.

NOTE: **The translate option has no effect since conversion of filenames between UNIX and filenames within a NPL diskimage is not performed. However, if the options are changed during the compiler display, conversion of filenames between those within NPL diskimages and UNIX filenames would be performed for the character pairs " " (NPL) and "_" (UNIX), and "/" (NPL) and "%" (UNIX).**

b2cb

```
b2c -display on -srcloc /dev/fd0 -objloc /dev/null -lstloc .
-lstformat .src -warnings off -translate 5F20.252F "$*"
```

The above batch file compiles input programs from a raw diskette in 720k or 1.44MB format, creating ASCII text files in the current directory, which may be edited and resubmitted to the compiler.

NOTE: **Conversion of filenames between those within the NPL diskimages and UNIX filenames is performed for the character pairs " " (NPL) and "_" (UNIX), and "/" (NPL) and "%" (UNIX) when creating the text files.**

b2cc

```
b2c -display on -rem$ on -objloc PLATTER1.BS2 -translate 5F20.252F "$*"
```

The above batch file compiles input programs in ASCII format from the current directory, creating p-code files in UNSCRAMBLED format which is placed in the diskimage file PLATTER1.BS2 also in the current directory, compiling any statements which start with a "REM $ PC", and displaying any compiler warning messages on the screen.

NOTE: The translate option is used when determining the name of the filename in the PLATTER1.BS2 diskimage, based on the .SRC files used as input.

b2cd

```
b2c -display on -lstloc /dev/fd048ds9 -objloc /dev/null -lstformat
2200 -warnings off -translate 5F20.252F "$*"
```

The above batch file compiles input programs in ASCII format from the current directory, creating output programs in 2200 atomized format directly on a 720K or 1.44MB diskette, which may then be ported over to the 2200.

NOTE: If the UNIX filenames contain "_" or "/" characters, the names of the files within the diskimage are automatically replace any "%" characters with the "/" character, and replace the "_" with a " ".

The use of the replaceable parameter ("$*") is not required since the DISPLAY ON option is specified. When the display option is specified, the compiler allows the entry of the input program name selections in the display despite the replaceable parameters. However, it does allow specification of the source program names on the command line and have them displayed automatically.

The Intel UNIX of the NPL supports the use of raw 360K diskettes. Refer to the NPL Programmer's Guide for details on the use of raw 360K diskettes on the Wang 2200.

9.5.2 Example Compiles

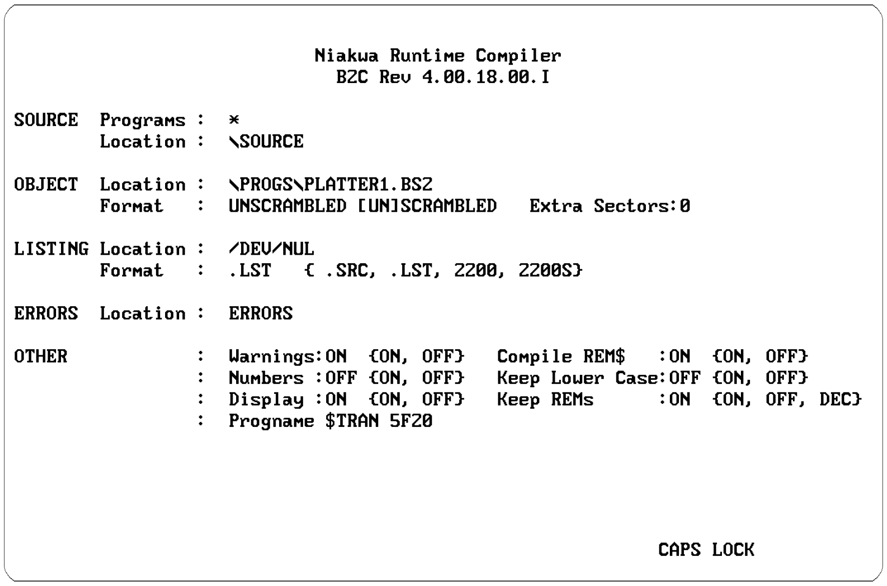
Chapter 14 of the NPL Programmer's Guide describes each option available for use when compiling programs. The remainder of this chapter provides several examples of commonly used compilations, indicating both the particular compiler command line and the compiler display. A brief discussion of what each compile is accomplishing, as well as an explanation of each compiler option is also included.

From a Compiled Diskimage to a Compiled Diskimage

Command line:

```
b2c -display on -srcloc /usr/PROGS/PLATTER1.BS2 -objloc
/usr/PROGS/PLATTER2.BS2 -errloc errors -rem$ on '*'
```

Display:



```
                              Niakwa Runtime Compiler
                                B2C Rev 4.00.18.00.I

SOURCE  Programs :  *
        Location :  \SOURCE

OBJECT  Location :  \PROGS\PLATTER1.BS2
        Format   :  UNSCRAMBLED [UN]SCRAMBLED   Extra Sectors:0

LISTING Location :  /DEV/NUL
        Format   :  .LST   { .SRC, .LST, 2200, 2200S}

ERRORS  Location :  ERRORS

OTHER            :  Warnings:ON  {ON, OFF}   Compile REM$   :ON  {ON, OFF}
                 :  Numbers :OFF {ON, OFF}   Keep Lower Case:OFF {ON, OFF}
                 :  Display :ON  {ON, OFF}   Keep REMs      :ON  {ON, OFF, DEC}
                 :  Progname $TRAN 5F20



                                                  CAPS LOCK
```

Discussion:

The above example indicates the procedure which would be used to create compiled programs in a diskimage file, directly from a diskimage file containing already compiled programs. With the KEEPREMS option set to OFF, REMs and spacing information are suppressed from the object programs. This would be useful for generating compressed program code which would then occupy less space on the end user's system. Unless warnings appear indicating that a statement requires a later revision of the RunTime, the code is considered compatible with all prior revisions of the RunTime. The diskimage file could then be copied onto diskette(s) using the NPL utilities and ported over to another machine operating under the same revision of the RunTime or an earlier release.

Explanation of Options:

SOURCE	Programs:	The "*" indicates that all programs in the specified srcloc are to be compiled.
	Location:	Indicates that the input programs are located in a diskimage file named PLATTER1.BS2, in the /usr/PROGS directory.
OBJECT	Location:	Indicates a diskimage file, PLATTER2.BS2, in the /usr/PROGS directory, which receives the compiled programs.
	Format:	The default UNSCRAMBLED indicates that program encryption does not take place.
	Extra Sectors:	The default 0 indicates that extra free sectors are not allocated to the programs.
LISTING	Location:	"/dev/null" indicates that no list output is generated.
	Format:	The .LST default is irrelevant since no list output is generated.
ERRORS	Location:	"errors" indicates the name of the file in the current directory which receives all warnings and errors during the compilation.
OTHER	Warnings:	ON indicates that all warnings encountered during the compile is displayed on the screen.
	Numbers:	The default OFF indicates that additional object code to keep track of multiple program statements is not generated.
	Display:	ON indicates that the compiler display screen is shown for review.
	Comp. REM$:	ON indicates that any statements beginning with REM $ PC has the program code contained in the statement compiled. The REM $ PC designation is then removed, due to KEEPREMS OFF.
	Keep L. Case:	Use of this compiler option is ignored under UNIX.
	Keep REMs:	The default OFF together with the REM$ option set to on, instructs the compiler to compile statements beginning with REM $ PC and then remove the REM $ PC designation from the statement. Also, the original format of certain literals (either HEX or quote string) may not be retained, and "special" program spacing is removed.

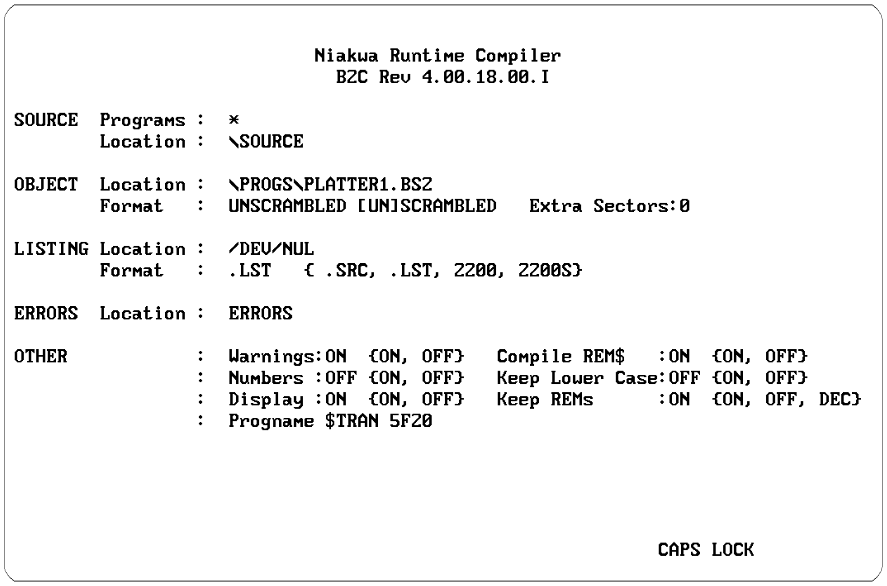
Prog. $TRAN: The default 5F20 is irrelevant since it only pertains to filename translations between programs stored as stand-alone UNIX files and programs stored in NPL diskimage files.

From a 2200 to a Compiled Diskimage

Command line:

```
b2c -display on -srcloc /dev/fd048ds9 -objloc /usr/PROGS/PLATTER1.BS2
-errloc errors '*'
```

Display:



```
                              Niakwa Runtime Compiler
                                B2C Rev 4.00.18.00.I

SOURCE  Programs :  *
        Location :  \SOURCE

OBJECT  Location :  \PROGS\PLATTER1.BS2
        Format   :  UNSCRAMBLED [UN]SCRAMBLED   Extra Sectors:0

LISTING Location :  /DEV/NUL
        Format   :  .LST   { .SRC, .LST, 2200, 2200S}

ERRORS  Location :  ERRORS

OTHER            :  Warnings:ON  {ON, OFF}   Compile REM$   :ON  {ON, OFF}
                 :  Numbers :OFF {ON, OFF}   Keep Lower Case:OFF {ON, OFF}
                 :  Display :ON  {ON, OFF}   Keep REMs      :ON  {ON, OFF, DEC}
                 :  Progname $TRAN 5F20




                                                          CAPS LOCK
```

Discussion:

The above example indicates the compilation procedure which would be used for the initial porting of programs from a Wang 2200 to a diskimage file on the system.

Explanation of Options:

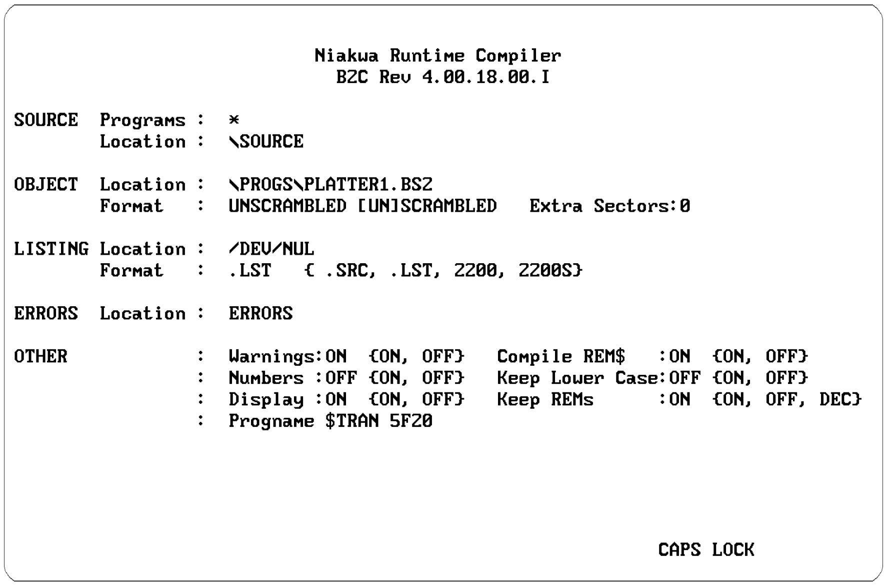
SOURCE	Programs:	The "*" indicates that all programs in the specified srcloc are to be compiled.
	Location:	/dev/fd048ds9 indicates that the input programs are located on a 360K raw format diskette in drive 0.
OBJECT	Location:	Indicates a diskimage file, PLATTER1.BS2 in the /usr/PROGS directory, will receive the compiled program.
	Format:	The default UNSCRAMBLED indicates that program encryption does not take place.
OBJECT	Extra Sectors:	The default 0 indicates that extra free sectors are not allocated to the programs.
LISTING	Location:	"/dev/null" indicates that no list output is generated.
	Format:	The .LST default is irrelevant since no list output is generated.
ERRORS	Location:	"errors" indicates the name of the file in the current directory which receives all warnings and errors during the compilation.
OTHER	Warnings:	ON indicates that all warnings encountered during the compile are displayed on the screen.
	Numbers:	The default OFF indicates that additional object code to keep track of multiple program statements is not generated.
	Display:	ON indicates that the compiler display screen is shown for review.
	Comp. REM$:	The default OFF indicates that any program statement beginning with REM $ PC is ignored by the compile.
	Keep L. Case:	Use of this compiler option is ignored under UNIX.
	Keep REMs:	The default OFF indicates that any program statement beginning with a REM is removed during the compile, the original format of certain literals (either HEX or quote string) may not be retained, and "special" program spacing is removed.
	Prog. $TRAN:	The default 5F20 is irrelevant since it only pertains to filename translations between programs stored as stand-alone UNIX files and programs stored in NPL diskimage files.

From a Compiled Diskimage to 2200 Atomized Code

Command line:

```
b2c -display on -srcloc /usr/PROGS/PLATTER1.BS2 -objloc /dev/null
-lstloc /dev/fd048ds9 -lstformat 2200 -errloc errors '*'
```

Display:



```
                              Niakwa Runtime Compiler
                                B2C Rev 4.00.18.00.I

SOURCE  Programs :  *
        Location :  \SOURCE

OBJECT  Location :  \PROGS\PLATTER1.BS2
        Format   :  UNSCRAMBLED [UN]SCRAMBLED   Extra Sectors:0

LISTING Location :  /DEV/NUL
        Format   :  .LST   { .SRC, .LST, 2200, 2200S}

ERRORS  Location :  ERRORS

OTHER            :  Warnings:ON  {ON, OFF}   Compile REM$    :ON  {ON, OFF}
                 :  Numbers :OFF {ON, OFF}   Keep Lower Case:OFF {ON, OFF}
                 :  Display :ON  {ON, OFF}   Keep REMs      :ON  {ON, OFF, DEC}
                 :  Progname $TRAN 5F20



                                                        CAPS LOCK
```

Discussion:

The above example indicates the procedure which would be used to create programs in 2200 atomized format, directly from a diskimage file containing already compiled programs. These programs could be created directly on a diskette in 360K raw format, as the example indicates, if they fit on one diskette. If not, then the LSTLOC should be a diskimage file on the hard disk, which could then be copied onto multiple diskettes using the NPL utilities, and ported over to the 2200.

Explanation of Options:

SOURCE	Programs:	The "*" indicates that all programs in the specified srcloc are to be compiled.
	Location:	Indicates that the input programs are located in a diskimage file named PLATTER1.BS2, in the /usr/PROGS directory.
OBJECT	Location:	"/dev/null" indicates that the compiler suppresses generating a second set of compiled programs.
	Format:	The default UNSCRAMBLED is irrelevant since an objloc of "/dev/null" is specified.
	Extra Sectors:	The default 0 is irrelevant since an objloc of "/dev/null" is specified.
LISTING	Location:	/dev/fd048ds9 indicates that the compiler generates output programs directly on a raw diskette in 360K format.
	Format:	The 2200 option indicates that the programs generated on the specified LSTLOC is in 2200 atomized format. Use of the 2200S option would additionally remove syntactically insignificant spaces from all programs, thus creating "compressed" program code.
ERRORS	Location:	"errors" indicates the name of the file in the current directory which receives all warnings and errors during the compilation.
OTHER	Warnings:	ON indicates that all warnings encountered during the compile are displayed on the screen. Warnings are not sent to the "lstloc" if the lstformat option is either 2200 or 2200S.
	Numbers:	The default OFF is irrelevant since an objloc of "/dev/null" is specified.
	Display:	ON indicates that the compiler display screen is shown for review.
	Comp. REM$:	The default OFF is irrelevant since an objloc of "/dev/null" is specified.
	Keep L. Case:	Use of this compiler option is ignored under UNIX.
	Keep REMs:	The default OFF is irrelevant since an objloc of "/dev/null" is specified.

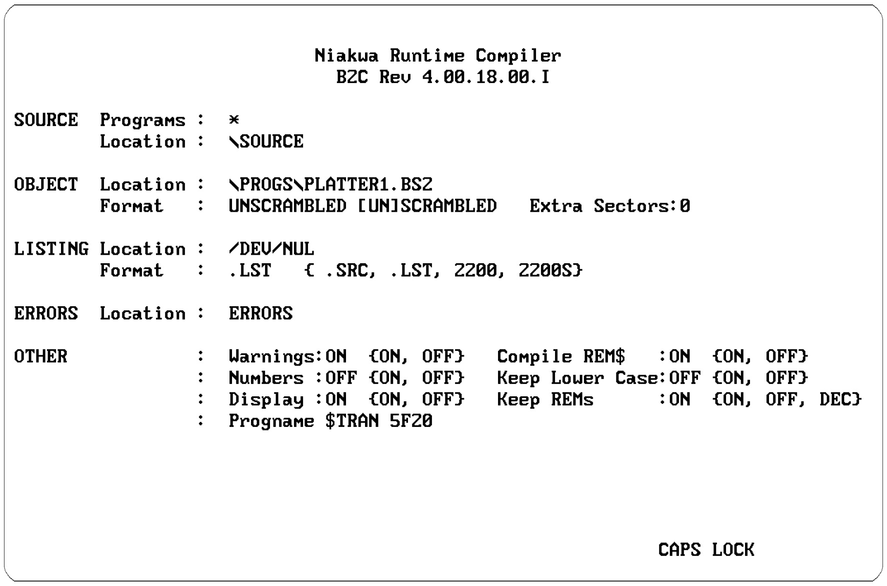
Prog. $TRAN: The default 5F20 is irrelevant since it only pertains to filename translations between programs stored as stand-alone UNIX files and programs stored in NPL diskimage files.

From a 2200 to a Compiled Diskimage and ASCII Text Files

Command line:

```
b2c -display on -srcloc /dev/fd048ds9 -objloc /usr/PROGS/PLATTER1.BS2
-lstloc /usr/SOURCE -lstformat .SRC -rem$ on -errloc errors '*'
```

Display:



```
                           Niakwa Runtime Compiler
                             B2C Rev 4.00.18.00.I

SOURCE  Programs :  *
        Location :  \SOURCE

OBJECT  Location :  \PROGS\PLATTER1.BS2
        Format   :  UNSCRAMBLED [UN]SCRAMBLED   Extra Sectors:0

LISTING Location :  /DEV/NUL
        Format   :  .LST   { .SRC, .LST, 2200, 2200S}

ERRORS  Location :  ERRORS

OTHER            :  Warnings:ON  {ON, OFF}   Compile REM$   :ON  {ON, OFF}
                 :  Numbers :OFF {ON, OFF}   Keep Lower Case:OFF {ON, OFF}
                 :  Display :ON  {ON, OFF}   Keep REMs      :ON  {ON, OFF, DEC}
                 :  Progname $TRAN 5F20



                                                      CAPS LOCK
```

Discussion:

The above example indicates the compilation procedure which would be used for the initial porting of programs from a Wang 2200 to Intel UNIX, creating both a diskimage file containing the compiled programs, and "source" code listings as ASCII text files in an UNIX directory.

Explanation of Options:

SOURCE	Programs:	The "*" indicates that all programs in the specified srcloc are to be compiled.
	Location:	/dev/fd048ds9 indicates that the input programs are located on a raw 360K format diskette in drive 0.
OBJECT	Location:	Indicates a diskimage file, PLATTER1.BS2 in the /usr/PROGS directory, which will receive the compiled programs.
	Format:	The default UNSCRAMBLED indicates that program encryption does not take place.
OBJECT	Extra Sectors:	The default 0 indicates that extra free sectors are not allocated to the programs.
LISTING	Location:	Indicates that source program listings are generated in the /usr/SOURCE directory.
	Format:	.SRC indicates that the programs generated in the specified lstloc are in ASCII text format.
ERRORS	Location:	"errors" indicates the name of the file in the current directory which receives all warnings and errors during the compilation.
OTHER	Warnings:	ON indicates that all warnings encountered during the compile are displayed on the screen and in the .SRC files.
	Numbers:	The default OFF indicates that additional object code to keep track of multiple program statements is not generated.
	Display:	ON indicates that the compiler display screen is shown for review.
	Comp. REM$:	The ON option indicates that any program statement beginning with REM $ PC has the program code contained in the statement compiled.
	Keep L. Case:	Use of this compiler option is ignored under UNIX.
	Keep REMs:	The default OFF, in conjunction with the REM$ option set to ON, instructs the compiler to compile statements beginning with REM $ PC, and then remove the REM $ PC designation from the program statement. Also, the orginal format of certain literals (either HEX or quote string) may not be retained, and "special" program spacing is removed.

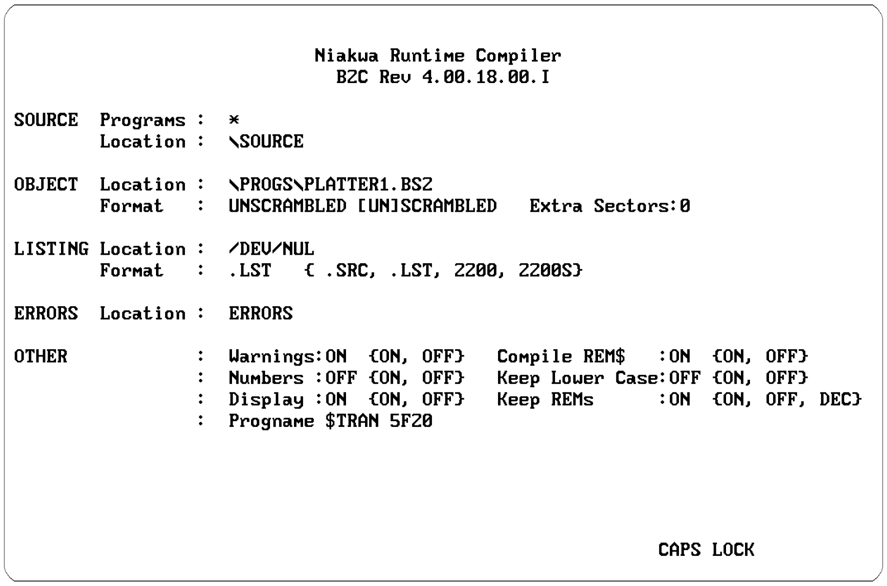
Prog. $TRAN: The default 5F20 indicates that spaces in input program names are converted to underline characters when creating ASCII text files in the specified lstloc.

From a Compiled Diskimage to ASCII Text Files

Command line:

```
b2c -display on -srcloc /usr/PROGS/PLATTER1.BS2 -objloc /dev/null
-lstloc /usr/SOURCE -lstformat .SRC -errloc errors '*'
```

Display:



```
                          Niakwa Runtime Compiler
                            B2C Rev 4.00.18.00.I

SOURCE  Programs :  *
        Location :  \SOURCE

OBJECT  Location :  \PROGS\PLATTER1.BS2
        Format   :  UNSCRAMBLED [UN]SCRAMBLED   Extra Sectors:0

LISTING Location :  /DEV/NUL
        Format   :  .LST   { .SRC, .LST, 2200, 2200S}

ERRORS  Location :  ERRORS

OTHER            :  Warnings:ON  {ON, OFF}   Compile REM$   :ON  {ON, OFF}
                 :  Numbers :OFF {ON, OFF}   Keep Lower Case:OFF {ON, OFF}
                 :  Display :ON  {ON, OFF}   Keep REMs      :ON  {ON, OFF, DEC}
                 :  Progname $TRAN 5F20



                                                          CAPS LOCK
```

Discussion:

The above example indicates the procedure which would be used to create "source" code program listings in ASCII text format in an UNIX directory, directly from a diskimage file containing already compiled programs.

Explanation of Options:

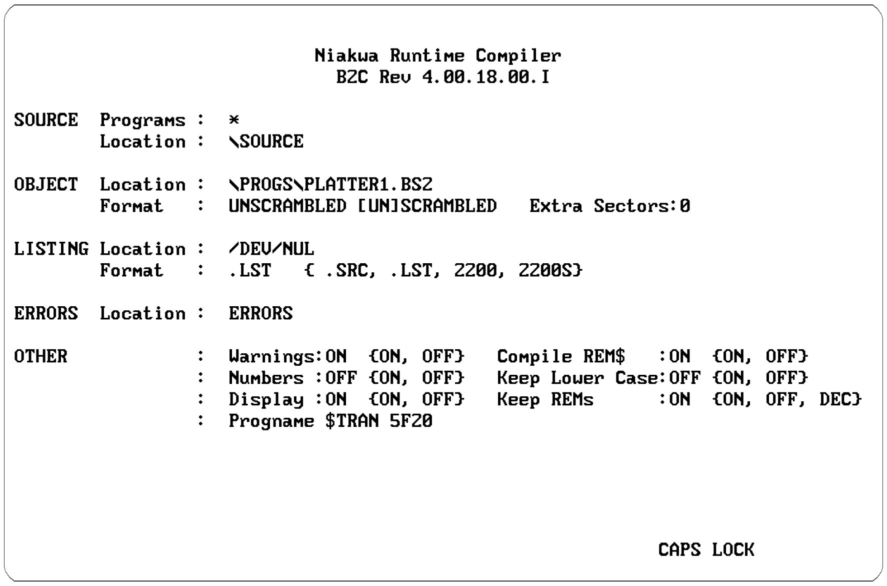
SOURCE	Programs:	The "*" indicates that all programs in the specified srcloc are to be compiled.
	Location:	Indicates that the input programs are located in a diskimage file named PLATTER1.BS2, in the /usr/PROGS directory.
OBJECT	Location:	"/dev/null" indicates that the compiler suppresses generating a second set of compiled programs.
	Format:	The default UNSCRAMBLED is irrelevant since an objloc of "/dev/null" is specified.
	Extra Sectors:	The default 0 is irrelevant since an objloc of "/dev/null" is specified.
LISTING	Location:	Indicates that source program listings are generated in the /usr/SOURCE directory.
	Format:	.SRC indicates that the programs generated in the specified lstloc are in ASCII text format.
ERRORS	Location:	"errors" indicates the name of the file in the current directory which receives all warnings and errors during the compilation.
OTHER	Warnings:	ON indicates that all warnings encountered during the compile are displayed on the screen and in the .SRC files.
	Numbers:	The default OFF is irrelevant since an objloc of "/dev/null" is specified.
	Display:	ON indicates that the compiler display screen is shown for review.
	Comp. REM$:	The default OFF is irrelevant since an objloc of "/dev/null" is specified.
	Keep L. Case:	Use of this compiler option is ignored under UNIX.
	Keep REMs:	The default OFF is irrelevant since an objloc of "/dev/null" is specified.
	Prog. $TRAN:	The default 5F20 indicates that spaces in input program names are converted to underline characters when creating ASCII text files in the specified lstloc.

From ASCII Text Files to Compiled Format in a Directory

Command line:

```
b2c -display on -srcloc /usr/AR/PROGS -objloc /usr/AR/PROGS -errloc
errors BOOT
```

Display:



```
                         Niakwa Runtime Compiler
                          B2C Rev 4.00.18.00.I

SOURCE  Programs :  *
        Location :  \SOURCE

OBJECT  Location :  \PROGS\PLATTER1.BS2
        Format   :  UNSCRAMBLED [UN]SCRAMBLED   Extra Sectors:0

LISTING Location :  /DEV/NUL
        Format   :  .LST   { .SRC, .LST, 2200, 2200S}

ERRORS  Location :  ERRORS

OTHER            :  Warnings:ON  {ON, OFF}   Compile REM$   :ON  {ON, OFF}
                 :  Numbers :OFF {ON, OFF}   Keep Lower Case:OFF {ON, OFF}
                 :  Display :ON  {ON, OFF}   Keep REMs      :ON  {ON, OFF, DEC}
                 :  Progname $TRAN 5F20




                                                     CAPS LOCK
```

Discussion:

The above example indicates the procedure which would be used to take a single "boot" program stored in ASCII text format, and compile it into a stand-alone .OBJ object file stored in an UNIX directory. Here is an example of a simple boot program:

```
10 $DEVICE(/D11)="/usr/AR/PROGS/PLATTER1.BS2"
20 $DEVICE(/D12)="/usr/AR/DATA/PLATTER1.BS2"
30 SELECT DISK D11
40 LOAD RUN"ARSTART"
```

Explanation of Options:

SOURCE	Programs:	Indicates that a single program named BOOT is to be compiled.
	Location:	Indicates that the input program is an ASCII text file with an extension of .SRC, and is located in the /usr/AR/PROGS directory.
OBJECT	Location:	Indicates that the compiler creates an object file with an extension of .OBJ in the /usr/AR/PROGS directory.
	Format:	The default UNSCRAMBLED indicates that program encryption does not take place.
	Extra Sectors:	The default 0 is irrelevant since a stand-alone .OBJ object file is being created.
LISTING	Location:	"/dev/null" indicates that no list output is generated.
	Format:	The .LST default is irrelevant since no output is generated.
ERRORS	Location:	"errors" indicates the name of the file in the current directory which receives all warnings and errors during the compilation.
OTHER	Warnings:	ON indicates that all warnings encountered during the compile are displayed on the screen.
	Numbers:	The default OFF indicates that additional object code to keep track of multiple program statements is not generated.
	Display:	ON indicates that the compiler display screen is shown for review.
	Comp. REM$:	The default OFF indicates that any program statement beginning with REM $ PC is ignored by the compile.
	Keep L. Case:	The default OFF is irrelevant since it only pertains to filename translations between programs stored as stand-alone UNIX files and programs stored in NPL diskimage files.
	Keep REMs:	The default OFF indicates that any program statement beginning with a REM is removed during the compile, the original format of certain literals (either HEX or quote string) may not be retained, and "special" program spacing is removed.

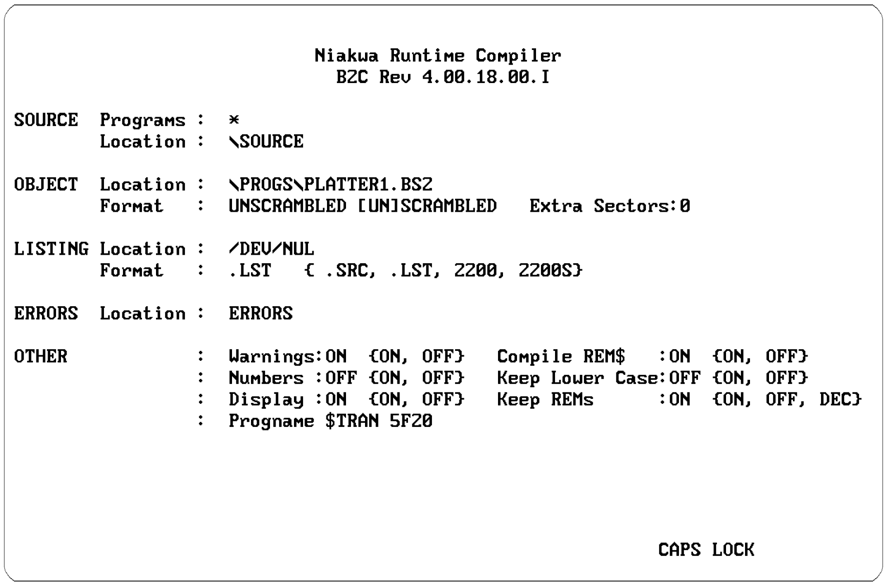
Prog. $TRAN:	The default 5F20 is irrelevant since it only pertains to filename translations between programs stored as stand-alone UNIX files and programs stored inNPL diskimage files.

From ASCII Text Files to a Compiled Diskimage

Command line:

```
b2c -display on -srcloc /usr/SOURCE -objloc /usr/PROGS/PLATTER1.BS2
-errloc errors -rem$ on -keeprems on *
```

Display:



```
                              Niakwa Runtime Compiler
                                B2C Rev 4.00.18.00.I

SOURCE  Programs :  *
        Location :  \SOURCE

OBJECT  Location :  \PROGS\PLATTER1.BS2
        Format   :  UNSCRAMBLED [UN]SCRAMBLED   Extra Sectors:0

LISTING Location :  /DEV/NUL
        Format   :  .LST   { .SRC, .LST, 2200, 2200S}

ERRORS  Location :  ERRORS

OTHER            :  Warnings:ON  {ON, OFF}   Compile REM$   :ON  {ON, OFF}
                 :  Numbers :OFF {ON, OFF}   Keep Lower Case:OFF {ON, OFF}
                 :  Display :ON  {ON, OFF}   Keep REMs      :ON  {ON, OFF, DEC}
                 :  Progname $TRAN 5F20




                                                        CAPS LOCK
```

Discussion:

The above example indicates the procedure which would be used to take input "source" programs stored in ASCII text format, and compile them into a diskimage file.

Explanation of Options:

SOURCE	Programs:	The "*" indicates that all programs in the specified srcloc are to be compiled.
	Location:	Indicates that the input programs are ASCII text files with an extension of .SRC, and are located in the /usr/SOURCE directory.
OBJECT	Location:	Indicates a diskimage file, PLATTER.BS2 in the /usr/PROGS directory, which will receive the compilied program.
	Format:	The default UNSCRAMBLED indicates that program encryption does not take place.
	Extra Sectors:	The default 0 indicates that extra free sectors are not allocated to the programs.
LISTING	Location:	"/dev/null" indicates that no list output is generated.
	Format:	The .LST default is irrelevant since no output is generated.
ERRORS	Location:	"errors" indicates the name of the file in the current directory which receives all warnings and errors during the compilation.
OTHER	Warnings:	ON indicates that all warnings encountered during the compile are displayed on the screen.
	Numbers:	The default OFF indicates that additional object code to keep track of multiple program statements is not generated.
	Display:	ON indicates that the compiler display screen is shown for review.
	Comp. REM$:	The ON option indicates that any program statement beginning with REM $ PC has the program code contained in the statement compiled.
	Keep L. Case:	The default OFF is irrelevant because of the use of the program name wildcard. If individual programs were specified and it was desirable to specify the case of the program name for the files generated in the diskimage, the lower case option should be set to ON.

Keep REMs:	The ON option indicates that REM statements and spacing information is retained in the object program. Also, the original format of certain literals (either HEX or quote string) is retained.
Prog. $TRAN:	The default value of 5F20 causes underline characters in stand-alone UNIX ".SRC" files to be converted to a space within the object diskimage.

CHAPTER 10

PORTING PROGRAMS AND DATA

10.1 Overview

This chapter provides an overview of porting NPL applications and data files to or from a UNIX system.

Section 10.2 discusses NPL specific options available for porting.

Section 10.3 discusses options for porting to/from specific environments outside the Run-Time.

NOTE: The discussions in this chapter are intended to provide general information on file transfer and porting options. For a complete description of the options mentioned in this chapter, refer to Chapter 15 of the NPL Programmer’s Guide.

10.2 Porting Options

Two options can be used for transfer of NPL applications and data on UNIX/Xenix systems. The following sections discuss these options.

10.2.1 Serial Communications

Direct file transfer from or to a UNIX system is possible using serial communications. For serial transfer of NPL programs and data to or from a UNIX system, any program that supports binary file transfer can be used. Many of these products also support terminal emulation. Be aware that the features used by NPL (downloadable fonts, local printer support, etc.) may not function properly on terminal emulation products. This can cause problems for NPL users.

Often the easiest method for transferring files serially between a UNIX system and another system is to use an MS-DOS system as an intermediate step. Most Intel-based UNIX systems are capable of reading MS-DOS format diskettes.

NOTE **Some communication programs may use keys that conflict with NPL key mapping. These conflicts can be avoided using the $KEYBOARD feature.**

Any file transfer product used must be capable of transferring eight-bit binary data.

10.2.2 "RAW" Diskette Transfer

The ability of NPL to access "raw" diskettes provides a means of transferring NPL diskimages from one machine to another regardless of the respective file formats of the native operating systems. A "raw" device is a physical disk which does not contain a native file system.

The UNIX implementation of NPL supports 5-1/4" 360K, 720K, and 1.2MB "raw" formatted diskettes, and 3-1/2" 720K, 1.44MB, and 2.88MB "raw" formatted diskettes. Most NPL-supported systems support one or more of the above formats. Refer to Appendix D for a list of all supported media on all NPL-supported machines.

Provided that diskette compatibility exists between two supported platforms, the following steps are required to port between a UNIX system and another supported platform.

1. Select a "raw" diskette format compatible with both the source and destination systems.

2. Use either the NPL Backup and Recovery Utilities or custom transfer utilities to transfer the programs and data between systems.

NOTE: Although the "raw" formatted diskettes are identical from one system to the next, the naming conventions of these diskettes may change from one operating system to the next.

For example:

MS-DOS

```
    $DEVICE(/D10)="A: 720=Y"          Defines drive A: as a "raw" 720K device.
    $DEVICE(/D10)="A: 1.44=Y"         Defines drive A: as a "raw" 1.44MB diskette.
```

SuperDOS

```
    $DEVICE(/D10)="1: 720=Y"          Defines drive 1: as a "raw" 720K device.
    $DEVICE(/D10)="1: 1.44=Y"         Defines drive 1: as a "raw" 1.44MB device.
```

UNIX

```
    $DEVICE(/D10)="/dev/fd1135ds9"    Defines drive 1 as a "raw" 720K device.
    $DEVICE(/D10)="/dev/fd1135ds18"   Defines drive 1 as a "raw" 1.44MB
                                      device.
    $DEVICE(/D10)="/dev/fd1135ds36"   Defines drive 1 as a "raw" 2.88MB
                                      device.
```

AIX on the IBM RS/6000

```
    $DEVICE(/D10)="/dev/fd0"          Defines drive 0 to read 720K or 1.44MB
                                      "raw" diskettes.
    $DEVICE(/D10)="/dev/fd0.9"
    $DEVICE(/D10)="/dev/fd0l"         Defines drive 0 to read/write 720K
                                      "raw" diskettes.
    $DEVICE(/D10)="/dev/fd0.18
    $DEVICE(/D10)="/dev/fd0h"         Defines drive 0 to read/write 1.44MB
                                      and 2.88MB diskettes.
```

Refer to Chapter 5 of the appropriate NPL operating system-specific supplement for more information on supported diskette naming conventions.

Refer to Chapter 15 of the NPL Programmer's Guide for more information on porting.

10.3 Specific Environments

The following section discusses the methods available to port NPL applications for UNIX from specific environments. Several of the methods described below require porting the application to an MS-DOS system first and then porting it to the UNIX environment.

10.3.1 From a Wang 2200

Direct transfer from the 2200 to UNIX file systems using serial communications is possible. However, Niakwa does not know of any existing software which does this. For now, the recommended method of porting from a 2200 to a UNIX filesystem is to use an IBM or compatible PC as an intermediate step. Refer to 10.3.2 for more information.

10.3.2 From an MS-DOS-based system

Some Intel-based UNIX systems are capable of reading and writing MS-DOS format diskettes. Niakwa programs and data can be ported from MS-DOS based systems to UNIX by copying them to diskette with the standard DOS copy commands, then reading them into the UNIX file system using the UNIX's dos copy (e.g., doscp) command. Refer to the UNIX system commands reference for more information.

10.3.3 From a SuperDOS-Based System

The simplest way to transfer files from a SuperDOS system to a UNIX system is to first transfer them to MS-DOS, then transfer to UNIX as described in Section 10.3.2 above.

The simplest way to transfer data between MS-DOS and SuperDOS is to use the PCFILE utility which is part of the SuperDOS operating system package. This program runs on SuperDOS and has a menu interface. However, PCFILE can only be used in SuperDOS real mode. SuperDOS 6.0 runs only in protected mode, and has a different DOS file transfer utility called PCXFER. For further details, refer to the SuperDOS Utilities Guide.

Alternatively, PC2200 by Computer Concepts transfers NPL data and programs between DOS and SuperDOS. It can receive as well as transmit between SuperDOS and an MS-DOS based system. This program emulates a Wang 2236/DW terminal. There are several considerations in using such a terminal with a SuperDOS system. Refer the NPL Super-DOS Supplement regarding Wang 2x36 terminal support under SuperDOS.

10.3.4 From Another UNIX Based System

The "tar" utility may be used to transfer programs or data between two UNIX based NPL systems. Diskimage files or other files related to the operation of the developer's software can be copied using the tar utility.

The following guidelines should be followed when using the tar utility:

- When restoring large files, be sure to set ulimit to 65000.
- Login as the super-user.
- Use of a blocking factor of 20 or greater substantially improves performance of the tar utility.

The following example shows how to copy the diskimage file /usr/BASIC2C/PLATTER1.BS2 using the tar utility from one UNIX system to another:

1. Login on the first machine as "root".
2. From the UNIX shell:

```
cd /
tar cvfbk /dev/rfd048ds9 20 360 /usr/BASIC2C/PLATTER1.BS2
```

copies diskimage file PLATTER1.BS2 from the /usr/BASIC2C/directory to one or more 360K diskettes.

or

```
cd /
tar cvfbk /dev/rfd096ds15 20 1200 /usr/BASIC2C/PLATTER1.BS2
```

copies diskimage file PLATTER1.BS2 from the /usr/BASIC2C directory on one or more 1.2MB diskettes.

NOTE: Other formats of diskette may be used by specifying the appropriate raw device name and diskette size (in kilobytes) on the appropriate parameters of the tar command.

3. Login to UNIX on the second machine as "root".

4. From the UNIX shell:

```
cd /
umask 0000     Set protection mode to full access for all users, if
               required.
ulimit 65000   Set maximum file creation size to 32 MB.
tar xvf /dev/fd048ds9
```

copies the diskimage file PLATTER1.BS2 from the 360K diskettes(s) to the /usr/BA-SIC2C directory.

or

```
tar xvf /dev/fd096ds15
```

copies the diskimage file PLATTER1.BS2 from the 1.2MB diskettes(s) to the /usr/BA-SIC2C directory.

NOTE: Refer to Chapter 15 in the NPL Programmer's Guide for more information on porting.

CHAPTER 11

MIXED LANGUAGE PROGRAMMING

11.1 Overview

The NPL External Subroutine Development Kit (BESDK), formerly Basic-2C, provides an interface to external subroutines written in other programming languages. However, there are both benefits and penalties which may occur as a result of using mixed language programming under NPL. The benefits include a potential increase in execution speed for selected processor-intensive functions, and the capability to access resources and features of a specific environment. The penalties include increased memory requirements, limited portability to other NPL environments, and a potentially less friendly environment for testing and error diagnosis.

This chapter concerns itself with the operating environment-specific features of the NPL External Subroutine Development Kit (BESDK) for UNIX and Xenix. For a complete discussion on the general operations of mixed language programming, refer to Chapter 16 of the NPL Programmer's Guide.

Section 11.2 discusses the contents of the BESDK.

Section 11.3 discusses the installation of the BESDK.

Section 11.4 discusses Xenix support.

Section 11.5 discusses Microsoft C under Xenix.

Section 11.6 discusses support of Microsoft Macro Assembler under Xenix.

Section 11.7 discusses UNIX support.

Section 11.8 discusses Microsoft C under UNIX V/386.

Section 11.9 discusses support of Microsoft Macro Assembler under UNIX.

Section 11.10 discusses flow control.

11.1.1 Differences from DOS/SuperDOS Releases

The MS-DOS and SuperDOS releases of NPL use the quick library mechanism of the Microsoft Linker, allowing the standard NPL RunTime program to load a specified set of routines after NPL starts. This approach does not work under the UNIX or Xenix environments where code, in general, cannot be dynamically linked at execution time.

Instead, the user subroutines must be linked with the NPL code itself, to produce an executable file which is a customized version of NPL with the user subroutines "built in." This is the approach used to support external subroutines under UNIX and Xenix.

Another major difference is that there are two versions of the NPL RunTime: one for standard Xenix (286) systems and a true UNIX (386) version that can only be used on approved 386/UNIX implementations. Therefore, there are two BESDK packages provided: one Xenix/286 BESDK for the standard Xenix version and one UNIX V/386 BESDK for the UNIX version.

NOTE: A custom RunTime made with the Xenix/286 BESDK is a 286 version. One made with the UNIX V/386 BESDK is a 386 version.

11.1.2 Choosing the Development Environment

When coding and linking external routines, the following implications must be considered:

- The operating system, BESDK version, and RunTime version the custom RunTime will work on.
- Because the user subroutines must be linked to produce a Xenix or UNIX binary executable file, the Xenix or UNIX software development system utilities, particularly the linker ("ld") and the C startup and support libraries, must have been installed on the development system.
- In addition to the issues relating to which version of the BESDK package to use on which operating system environment, there are issues relating to the actual coding of the external routines that depend on which BESDK package is used. These issues are explained in detail below in the specific sections on Xenix and UNIX support.

NOTE: Some Xenix system calls are not supported by UNIX and vice versa. To ensure portability between systems, use only those system calls that are common to both Xenix and UNIX.

Some UNIX V/386 systems may be capable of running Xenix large model binary executable files, but the software development tools may not support cross-development of Xenix executable files. Production of Xenix executable files on a UNIX system requires installation and modification of the Xenix/286 BESDK files. If UNIX executable files are also being produced, then the Xenix/286 and UNIX/386 sets of BESDK files have to be installed in separate directories on the UNIX system. Because of the complexities involved, it is strongly recommended that you do not attempt to produce Xenix executable files on UNIX systems. All compiling and linking should be done on the same type of system that you intend the executable file to run on.

11.1.3 Security

Any custom version of rti or rtp produced by using the BESDK procedures, although not physically copy protected, does not execute unless a standard RunTime is installed on the system where the custom version is to be used.

NOTE: The standard RunTime installed must be of the same type as the custom RunTime. That is, a standard Xenix RunTime must be installed for custom RunTimes created using the Xenix/286 BESDK and a standard UNIX RunTime must be installed for custom RunTimes created with UNIX V/386 BESDK. In addition, the revision level of the installed standard RunTime must be equal to or greater than the revision level of the custom RunTime.

When executed, the custom RunTime extracts the Serial Number from the installed "standard" RunTime and uses the Serial Number to perform the security check. If the Gold Key security from the standard RunTime is installed on the system, the security check is passed based on that. Otherwise, the Gold Key security check is performed. If this is needed, the Gold Key from the installed standard RunTime must be used.

NOTE: The custom RunTime also extracts and enforces the user limit from the standard RunTime installed on the system.

Installation of the Custom RunTime at End-User Sites

The procedure for installation of custom RunTimes at end-user sites is simple:

1. Install a standard RunTime of the same type (Xenix or UNIX) and version (or greater) as the custom RunTime on the end user's system using the normal installation procedure.

2. Copy the custom RunTime produced on the development system to the end user's machine.

11.1.4 Upgrades

When new releases of NPL become available, updates of the libraries used to make the customized versions of NPL will also be made available. Periodically, releases of the libraries with bug corrections corresponding to interim patches may also be made. It is the developer's responsibility to produce new versions of their customized RunTime programs and distribute them to customers.

NOTE: For formal upgrades of the RunTime, the standard RunTime at end user sites must be upgraded before installing an upgraded custom RunTime.

11.2 Contents of the BESDK

As explained above, there are two BESDK packages provided with the NPL Development Package. The first package, Xenix/286 BESDK, is contained on a single diskette and is intended for use within a Xenix operating environment. The second package, UNIX V/386 BESDK, is contained on two diskettes and is intended for use within a UNIX System V/386 operating environment. The BESDK diskettes are stored in UNIX/Xenix tape archive ("tar") format, and must be installed before they can be used. Within each BESDK package, files within the archive are separated into directories, each of which illustrates an example of linking an external subroutine in a particular environment using a particular language. The function performed by the subroutine is the same in each case, and is analogous to the Microsoft C example followed in the text.

The examples are provided mainly to test normal versions of compilers, linkers, etc.; that are being used with the source files that have been pretested, and to help clarify any points that may be unclear in the text.

HINT: It is recommended that example standalone and customized versions of the BESDK example be produced before starting a customized project, to ensure the various utilities work together as they should.

The contents of the BESDK are listed below:

NOTE: The names of the files contained on the Xenix BESDK are identical to those contained on the UNIX BESDK. The only difference between the two is the name of the root directory on the archive. The root directory on Xenix/286 BESDK is named /xextrn; whereas, the UNIX V/386 BESDK root is named /uextrn.

/xextrn	Xenix/286 BESDK.
/uextrn	UNIX/386 BESDK.

Contains all subdirectories pertaining to the operation of BESDK. The only file in this directory is the "readme" file. This file may contain amendments to existing documentation or additional information not available at press time. It is advisable to read this document before using BESDK.

The remaining files are separated into the following subdirectories under /xextrn or /uextrn directories respectively:

/bin	Contains two batch command files that can be used to link a set of external subroutines with either rti or rtp:
makertix	Links the specified object files with NPL to create an Interpretive RunTime with external subroutines (rtix).
makertpx	Links the specified object files with NPL to create a Non-Interpretive RunTime with external subroutines (rtpx).
/include	This directory contains files that are common to all implementations or to all implementations of a specific language. It is recommended that you do not change the files in this directory. The files provided in this directory are:
MYBOOT.SRC	An NPL source file, which performs a simple test of the example external subroutine.
MYBOOT.OBJ	Compiled version of the MYBOOT.SRC boot program to test the example external subroutines and FUNCTIONs. MYBOOT contains only configuration commands and loads MYSTART form the MYMODULE.BS2 diskimage.
MYSTART.SRC	Source version of the NPL program used to test the example subroutines and FUNCTIONs.
MYMODULE.SRC	Source version of the NPL library module used to specify the interface to the example subroutines and FUNCTIONs and containing a sample CALLBACK function.
MYMODULE.BS2	Compiled version of the MYMODULE.SRC and the MYSTART.SRC programs in a diskimage.

makefile	A make script that compiles MYBOOT.OBJ and MYMODULE.BS2. This assumes that the necessary source files (MYBOOT.SRC, MYSTART.SRC, MYMODULE.SRC) and the diskimage MYMODULE.BS2 are in the current directory, and that b2c is accessible on the PATH. To use, type "make".
rtpall.h	Standard include file for C programs, containing the structure and type definitions needed to write C code for the BESDK. The Release IV version of this file is expanded and contains a number of macros and types that did not exist in Release III.
rtpall.inc	For programmers that have BESDK libraries in Microsoft Assembler. Release IV features are not supported under this language.
rtpdeffn.h	Operating system dependent macros to define the attributes of external routines.
/lib	Contains the object files for NPL in archive library form.
/noextern	Contains two files for creating a RunTime without external subroutines; these are provided to allow a simple test of the linking procedure.
noextern	Object file containing "_main" and "_RTPEXT" routines that have no external subroutines.
makefile	Make file to link "noextern.o" with NPL and create an Interpretive RunTime "rtin". To use, select yourself to this directory and enter "make rtin".
/cexam	Contains example files for Xenix/UNIX implementations of external subroutines using C.
/mexam	Contains example files for Xenix/UNIX implementations of external subroutines using MASM.

In each of the example directories, the following files are provided:

mymain.x	Source file for example mainline.
myrtp.x	Source file for example rtp test subroutine.
myrtpext.x	Source file for example RTPEXT subroutine.
mysub.x	Source file for example DEFFN' subroutine.

where x =
c for C programs
s for MASM programs

makefile	Script for "make" utility to produce both mainline, customized rti and customized rtp. To run it, enter any of the following commands:
"make mymain"	Compiles and links example mainline.
"make rtix"	Compiles & links example customized rti (Interpretive), called "rtix".
"make rtpx"	Compiles & links example customized rtp (Non-Interpretive), called "rtpx".

NOTE: The makefile in the /mexam directory comes in two versions, named makefile.altos and makefile.sco. The two are functionally identical, except that makefile.sco contains an additional command to convert object files from OVF format to Common Object Format, which is required by most UNIX/386 environments. Depending on the system needs, choose the appropriate makefile, and rename it (or copy or link it) to "makefile". (e.g. "cp makefile.sco makefile")

The /cexam directory also contains the following files, specifically for Release IV callback features:

mycallbk.c	Source code to illustrate the use of a C function (mykeyin) that performs a callback to the NPL function 'CallBack-Keyin.

mycallbk.h	Include file containing the parameter block specifications required by mycallbk.c
myproc.c	Source code to illustrate the implementation of the example external PROCEDURE in C.
myproc.h	Include file containing the parameter block specifications required by myproc.c.

NOTE: If the makefile itself is changed, delete all previously made .o files in the directory before running make again.

The "makefile" make scripts assume:

- The compiler executables (such as cc, masm, ld, b2c, etc.) that may be required can be accessed (the environment variable PATH is set to allow these to be found),
- The example source files and makefile are in the current directory.
- The example include directory can be accessed as "../include".
- The command files "makertix" and "makertpx" can be found in "../bin".
- All required system libraries are in the default directory assumed by the standard utilities (/usr/lib).

The make script files assume the current (4.10 or later) version of the NPL compiler "b2c" is on the execution PATH.

11.3 Installation of the BESDK Diskette

The BESDK diskette(s) have been produced in UNIX/Xenix Archive format on two different media: 5-1/4" 1.2MB diskettes and 3-1/2" 720K diskettes. This is commonly known as "tar" format.

To install the BESDK diskette(s), select the directory to install the BESDK in and enter the following command for the appropriate media type you are installing (this command must be entered for each diskette):

1.2MB media:

```
tar xvf /dev/fd096ds15
```

720K media:

```
tar xvf /dev/fd135ds9
```

The BESDK directories and files are extracted to the currently selected directory.

11.4 Xenix Support

External subroutines are supported for Xenix in the native environment. Compiling, linking and running customized versions of rti and rtp under Xenix can be executed provided access to the Xenix software development system is available.

The entry point of the program must be labeled "__start" (two underlines) and must be a PUBLIC symbol. Linking with the standard C startup and support library meets this condition.

11.5 Microsoft C under Xenix

As indicated in the example, writing external subroutines in Microsoft C for Xenix is relatively straightforward. Examples assume SCO Xenix 3.2 or later. Earlier versions may also work but are not tested.

NOTE: The C compiler adds an underscore ("_") to the start of all labels. In the following discussion, labels are presented the way they must appear in the source files of the C routines. For example, the RTPEXT routine must be defined with the name "RTPEXT", and not "_RTPEXT". When it is compiled, the resulting object file contains the symbol "_RTPEXT". This fact is mentioned here in case it is necessary to examine the object files.

11.5.1 General

Use the large model 286 with extensions (-M2le) option on all "cc" compile commands, or ensure that this is part of the COPTS make variable.

Make sure the include files provided with the BESDK are available either in a directory specified by a "-Idirectory" option to the "cc" command or as part of the COPTS make variable.

11.5.2 Mainline

The starting label of user code is called "main".

NOTE: Substantial startup code from the C library is executed before reaching the "main" routine. The standard entry point of an image linked using Microsoft C is "__start", a library routine.

The RTP subroutine should be referenced as an external with the C naming conventions.

11.5.3 Calling Conventions for BESDK Subroutines

Test rtp Subroutines

Declare all GOSUB' routines using "pascal" and "far" calling conventions. The "rtpdeffn.h" include file for Xenix defines "rtpdeffn_ext" as equivalent to this designation.

RTPEXT Subroutine

The RTPEXT subroutine should be defined as a procedure with the C naming conventions. When called, the address of the rtpdef structure (defined in the include file rtpall.h) is the only parameter. The first field of this structure is a rtpreq structure (defined in the include file rtpall.h).

GOSUB' Subroutines:

Use the "pascal" designation of declarations of all subroutines that are called using the GOSUB' interface. The "rtpdeffn.h" include file for Xenix defines "rtpdeffn_ext" as equivalent to this designation.

Formats of strings in NPL do not have a zero-terminator and are not variable length. If strings are to be used by C library routines, you must make copies which have trailing spaces removed and a zero terminator added.

11.5.4 Linkage of Test Program

If the "cc" command is provided with the names of all source files (or .o files produced on previous compiles), it automatically invokes the linker with appropriate options (unless the -c option is specified) to produce the standalone test program. There is no need to specify which libraries are to be used by the compiler, provided the appropriately named libraries can be found in the standard system library directory (/usr/lib).

The files required for production of the standalone should include:

- The mainline
- The RTP test subroutine
- The RTPEXT subroutine (optional but recommended)
- The GOSUB' subroutines
- Any FUNCTION or PROCEDURE subroutines. If these are used, the rtpparm.o module must also be included.

Implied in the linkage are:

- The C support library Llibc.a (name may vary)
- The C segment module Lseg.o (may be in support library)
- The C startup module Lcrt0.o (may be in support library)

11.5.5 Linkage of Customized rti or rtp

If the "cc" command is provided with the names of all source files (or .o files produced on previous compiles) and the provided support libraries in the required order, it automatically invokes the linker with appropriate options (unless the -c option is specified) to produce the customized rti or rtp program. All libraries to be used by the compiler must be specified, except those located in the standard system library directory (/usr/lib). The resultant file name should be specified to indicate a variant version of rti or rtp, e.g. "-o rtix".

On some systems the linker may generate a "too many public symbols" warning. This is caused by the large size of NPL, and can be safely ignored.

The files required for production of the customized rti or rtp should include:

- ../lib/start (from BESDK)
- The mainline
- The RTPEXT subroutine
- The GOSUB' subroutines
- Any FUNCTION or PROCEDURE subroutines. If these are used, the rtpparm.o module must also be included.
- ../lib/rtplib (from BESDK)
- ../lib/rtpblib (Non-Interpretive rtp only)
- ../lib/srclib (Interpretive rti only)
- ../lib/srcblib (Interpretive rti only)
- ../lib/last (from BESDK)

Implied in the linkage are:

- The C support library Llibc.a (name may vary)
- The C segment module Lseg.o (may be in support library)
- The C startup module Lcrt0.o (may be in support library)

For example:

```
cc -Ml ../lib/start mymain.o myrtpext.o mysub.o ../lib/rtplib \
 ../lib/rtpblib ../lib/last -o rtpx
```

produces "rtpx" (Non-Interpretive RunTime, with extensions).

```
cc -Ml ../lib/start mymain.o myrtpext.o mysub.o ../lib/rtplib \
 ../lib/srclib ../lib/srcblib ../lib/last -o rtix
```

produces "rtix" (Interpretive RunTime, with extensions).

The batch command files supplied on the BESDK diskette may also be used to simplify the above commands to the following:

```
../bin/makertpx mymain.o myrtpext.o mysub.o
```

produces "rtpx" (Non-Interpretive RunTime, with extensions).

and

```
../bin/makertix mymain.o myrtpext.o mysub.o
```

produces "rtix" (Interpretive RunTime, with extensions).

11.6 Microsoft Macro Assembler Under Xenix

External subroutines and the mainline can be written entirely in MASM Macro assembler if required. Macro assembler has the advantage that support code dragged in from the libraries is usually minimal, and so the resulting library is often more compact than if written in a high-level language. However, the code is generally much more difficult to write and less portable when complete.

Microsoft masm may not be included in the software development package for all Xenix/286 systems. Examples assume Microsoft MASM ver 4.00 or later. Earlier versions may also work but are not tested.

External libraries written in Macro Assembler do not support the FUNCTION or PROCEDURE interfaces or callbacks to NPL.

11.6.1 General

Make sure the include files provided with the BESDK are available in a directory specified by the "-Idirectory" option to the "masm" command.

If a module refers to an external routine which is located in a library, linkage is simplified by adding an INCLUDELIB statement to the module referring to the required library.

11.6.2 Mainline

While it is possible to write an entire standalone module in macro assembler, interface to the Xenix kernel on most systems is not supported unless the standard C segmentation and startup modules are the first two modules linked. In this case, the entry point of the assembly mainline must be labeled "_main" (one underline) and must be a PUBLIC symbol. The module containing this label should be assembled with the "Mx" option to ensure that the lower case label is not translated by the assembly to upper case. The entry point _main is assumed by the C startup module, and must have this name.

The rtp subroutine should be referenced as a far external with the name "_RTP".

Use Microsoft conventions for segment names. Explicit segment names must be used. Be sure that:

- All code segments have combine class "CODE".
- All near-data segments (and standard stack) have combine class "DATA", and are part of the group named DGROUP.

The module must not depend on any specific segment ordering. Segment ordering is defined by the C segmentation module (i.e., all "CODE" first, DGROUP last, others in between). DGROUP segments of class "BSS" and "STACK" are always moved to the end of DGROUP. External symbols "_edata" and "_end" are defined by the linker to be offset DGROUP:BSS and offset DGROUP:STACK respectively.

At entry point _main, SS:SP are set to the system stack area. Do not attempt to use a different stack area.

If additional memory is required, use the system routines (malloc() or sbrk()) to obtain a pointer to new memory. Freeing memory by using brk() or sbrk() is usually inadvisable under Xenix.

Some additional restrictions apply (because code is running in protected mode at privilege level 3) that should be noted by those used to programming in real mode. These include:

- Segment addresses are no longer directly related to physical addresses. Operations which use this dependence, (e.g. to address huge data objects) will not work.

- Modifications to code segments are prohibited.
- Execution of code located in data segments is prohibited.
- Privileged instructions may not be used (e.g., port I/O, CLI/STI).

11.6.3 Calling Conventions for BESDK Subroutines

Test rtp Subroutines

Declare rtp subroutine as public with name "_RTP".

Call GOSUB' subroutines using pascal calling conventions (push arguments in order used in GOSUB' statements, assume arguments popped by subroutine).

RTPEXT Subroutine:

The RTPEXT subroutine should be defined as a far procedure with the name "_RTPEXT". When called, the address of the RTPDEF structure (defined in the include file rtpall.inc) is on the stack as a far pointer. The first field of this structure is an RTPREQ structure (defined in the include file rtpall.inc).

GOSUB' Subroutines

Each subroutine should be defined as a far procedure. When called, the parameters are on the stack below the return address. The first parameter of the GOSUB' is pushed first and the last parameter pushed last.

A string parameter is passed as:

```
PUSH                SEG <string>
PUSH                OFFSET <string>
PUSH                SIZE  <string>
```

A numeric parameter is passed as:

```
PUSH                SEG <rtpnum structure>
PUSH                OFFSET <rtpnum structure>
```

The rtpnum structure is defined in the include file rtpall.inc.

To conform to pascal calling conventions, use the form of the "RET" instruction that automatically pops parameters from the stack (4 bytes per numeric parameter + 6 bytes per string parameter).

11.6.4 Linkage of Test Program

Programs written in Macro Assembler must be linked to produce an executable file. All input files to "ld" must be the result of previously run assemblies or libraries of files.

HINT: Although "ld" can be used to invoke the linker directly, it is easier to invoke it by using the "cc" command. The advantage of this is that "cc" automatically links in the C startup and support libraries which are required so they do not have to be named explicitly, provided the appropriately named libraries can be found in the standard system library directory (usr/lib).

On some systems, the linker may generate a "too many public symbols" warning. This is caused by the large size of NPL, and can be safely ignored.

The files required for production of the standalone should include:

- The mainline (e.g. mymain.o)
- The RTP test subroutine (e.g. myrtp.o)
- The RTPEXT subroutine (optional but recommended)
- The GOSUB' subroutines (e.g. mysub.o)

Implied in the linkage are:

- The C support library Llibc.a (name may vary)
- The C segment module Lseg.o (may be in support library)
- The C startup module Lcrt0.o (may be in support library)

For example:

```
cc -Ml -o mymain mymain.o myrtp.o myrtpext.o mysub.o
```

11.6.5 Linkage of Customized rti or rtp

As stated in the previous section, "cc" can be used to invoke the linker and automatically link the required C startup routines and support libraries.

The files required for production of the customized rti or rtp should include:

- ../lib/start (from BESDK)
- The mainline
- The RTPEXT subroutine
- The GOSUB’ subroutines
- ../lib/rtplib (from BESDK)
- ../lib/rtpblib (Non-Interpretive rtp only)
- ../lib/srclib (Interpretive rti only)
- ../lib/srcblib (Interpretive rti only)
- ../lib/last (from BESDK)

Implied in the linkage are:

- The C support library Llibc.a (name may vary)
- The C segment module Lseg.o (may be in support library)
- The C startup module Lcrt0.o (may be in support library)

For example:

```
cc -Ml ../lib/start mymain.o myrtpext.o mysub.o ../lib rtplib \
  ../lib/rtpblib ../lib/last -o rtpx
```

produces "rtpx" (Non-Interpretive RunTime, with extensions).

```
cc -Ml ../lib/start mymain.o myrtpext.o mysub.o ../lib/rtplib \
  ../lib/srclib ../lib/srcblib ../lib/last -o rtix
```

produces "rtix" (Interpretive RunTime, with extensions).

The batch command files supplied on the BESDK diskette may also be used to simplify the above commands to the following:

```
../bin/makertpx mymain.o myrtpext.o mysub.o
```

produces "rtpx" (Non-Interpretive RunTime, with extensions).

and

```
    ../bin/makertix mymain.o myrtpext.o mysub.o
```

produces "rtix" (Interpretive RunTime, with extensions).

11.7 UNIX Support

External subroutines are supported for UNIX V/386 in the native environment. Compiling, linking and running customized versions of rti and rtp under UNIX V/386 can be executed provided access to the UNIX software development system is provided

The entry point of the program must be labeled "_start" (one underline) and must be a PUBLIC symbol. Linking with the standard C startup and support library meets this condition.

11.8 Microsoft C under UNIX

As indicated in the example, writing external subroutines in Microsoft C for UNIX V/386 is relatively straightforward. Examples assume Altos UNIX System V5.3bT1 or later, or SCO UNIX version 3.2 or later. Earlier versions may also work but are not tested.

11.8.1 General

No special models are used with UNIX V/386 - all addresses are 32 bit offsets (segmented addresses are not generally used by applications under UNIX V/386).

Make sure the include files provided with the BESDK are available either in a directory specified by a "-Idirectory" option to the "cc" command or as part of the COPTS make variable.

11.8.2 Mainline

The starting label of user code is called "main".

NOTE: Substantial startup code from the C library is executed before reaching the "main" routine. The standard entry point of an image linked using Microsoft C is "_start", a library routine.

The RTP subroutine should be referenced as an external with the C naming conventions.

11.8.3 Calling Conventions for BESDK Subroutines

Test rtp Subroutines

Declare all GOSUB' routines with the parameters listed in reverse order. The "rtpdeffn.h" include file for UNIX defines "rtpdeffn_ext" as equivalent to a null symbol for this implementation.

RTPEXT Subroutine

The RTPEXT subroutine should be defined as a procedure with the C naming conventions. When called, the address of the rtpdef structure (defined in the include file rtpall.h) is the only parameter. The first field of this structure is a rtpreq structure (defined in the include file rtpall.h).

GOSUB' Subroutines

All subroutines called using the GOSUB' interface must be declared with the parameters listed in reverse order. The "rtpdeffn.h" include file for UNIX defines "rtpdeffn_ext" and "rtpdeffn_ptr" as equivalent to null symbols for this implementation.

Formats of strings in NPL do not have a zero-terminator and are not variable length. If strings are to be used by C library routines, you must make copies which have trailing spaces removed and a zero terminator added.

11.8.4 Linkage of Test Program

If the "cc" command is provided with the names of all source files (or .o files produced on previous compiles), it automatically invokes the linker with appropriate options (unless the -c option is specified) to produce the standalone test program. There is no need to specify which libraries are to be used by the compiler, provided the appropriately named libraries can be found in the standard system library directory (/usr/lib).

The files required for production of the standalone should include:

- The mainline

- The RTP test subroutine
- The RTPEXT subroutine (optional but recommended)
- The GOSUB' subroutines
- Any FUNCTION or PROCEDURE subroutines. If these are used, the rtpparm.o module must also be included.

Implied in the linkage are:

- The C startup module crt1.o
- The C ending module crtn.o

11.8.5 Linkage of Customized rti or rtp

If the "cc" command is provided with the names of all source files (or .o files produced on previous compiles) and the provided support libraries in the required order, it automatically invokes the linker with appropriate options (unless the -c option is specified) to produce the customized rti or rtp program. All libraries to be used by the compiler must be specified, except those located in the standard system library directory (/usr/lib). The resultant file name should also be specified to indicate a variant version of rti or rtp, e.g. "-o rtix".

The files required for production of the customized rti or rtp should include:

- ../lib/start (from BESDK)
- The mainline
- The RTPEXT subroutine
- The GOSUB' subroutines
- Any FUNCTION or PROCEDURE subroutines. If these are used, the rtpparm.o module must also be included.
- ../lib/rtplib (from BESDK)

- ../lib/rtpblib (Non-Interpretive rtp only)
- ../lib/srclib (Interpretive rti only)
- ../lib/srcblib (Interpretive rti only)
- ../lib/last (from BESDK)

Implied in the linkage are:

- The C startup module crt1.o
- The C ending module crtn.o

For example:

```
cc ../lib/start mymain.o myrtpext.o mysub.o ../lib/rtplib \
  ../lib/rtpblib ../lib/last -o rtpx
```

produces "rtpx" (Non-Interpretive RunTime, with extensions).

```
cc ../lib/start mymain.o myrtpext.o mysub.o ../lib/rtplib \
  ../lib/srclib ../lib/srcblib ../lib/last -o rtix
```

produces "rtix" (Interpretive runTime, with extensions).

The batch command files supplied on the BESDK diskette may also be used to simplify the above commands to the following:

```
../bin/makertpx mymain.o myrtpext.o mysub.o
```

produces "rtpx" (Non-Interpretive RunTime, with extensions).

and

```
../bin/makertix mymain.o myrtpext.o mysub.o
```

produces "rtix" (Interpretive RunTime, with extensions).

11.9 Microsoft Macro Assembler Under UNIX

External subroutines and the mainline can be written entirely in Macro assembler if required. Macro assembler has the advantage that support code dragged in from the libraries is usually minimal, and so the resulting library is often more compact than if written in a high-level language. However, the code is generally much more difficult to write and less portable when complete.

Microsoft masm may not be included in the software development package for all UNIX V/386 systems. Examples assume Microsoft MASM ver 4.00 or later. Earlier versions may also work but are not tested.

External libraries written in Macro Assembler do not support the FUNCTION or PROCEDURE interfaces or callbacks to NPL.

11.9.1 General

Make sure the include files provided with the BESDK are available in a directory specified by the "-Idirectory" option to the "masm" command.

If a module refers to an external routine which is located in a library, linkage is simplified by adding an INCLUDELIB statement to the module referring to the required library.

11.9.2 Mainline

While it is possible to write an entire standalone module in macro assembler, interface to the UNIX kernel on most systems is not supported unless the standard C startup and end modules are the first and last modules linked. In this case, the entry point of the assembly mainline must be labeled "_main" (one underline) and must be a PUBLIC symbol. The module containing this label should be assembled with the "Mx" option to ensure that the lower case label is not translated by the assembly to upper case. The entry point _main is assumed by the C startup module, and must have this name.

The rtp subroutine should be referenced as a near external.

Use Microsoft conventions for small model segment names. Explicit segment names must be used. Be sure that:

- All code segments have combine class "CODE".
- All data segments (and standard stack) have combine class "DATA", and are part of the group named DGROUP.

At entry point _main, SS:SP are set to the system stack area. Do not attempt to use a different stack area.

All segments should have a 32-bit wordsize (USE32 directive).

Avoid using the segment registers.

If additional memory is required, use the system routines (malloc() or sbrk()) to obtain a pointer to new memory. Freeing memory by using brk() or sbrk() is usually inadvisable under UNIX.

Some additional restrictions apply (because code is running in protected mode at privilege level 3) that should be noted by those used to programming in real mode:

- Modifications to code segments are prohibited.
- Execution of code located in data segments is prohibited.
- Privileged instructions may not be used (e.g. port I/O, CLI/STI).

11.9.3 Calling Conventions for BESDK Subroutines

Test rtp Subroutines

Declare rtp subroutine as public.

Call GOSUB' subroutines using pascal calling conventions (push arguments in order used in GOSUB' statements, assume arguments popped by subroutine).

RTPEXT Subroutine

The RTPEXT subroutine should be defined as a near procedure. When called, the address of the RTPDEF structure (defined in the include file rtpall.inc) is on the stack as a near pointer. The first field of this structure is an RTPREQ structure (defined in the include file rtpall.inc).

GOSUB’ Subroutines

Each subroutine should be defined as a near procedure. When called, the parameters are on the stack below the return address. The first parameter of the GOSUB’ is pushed first and the last parameter pushed last.

A string parameter is passed as:

```
PUSH              OFFSET <string>
PUSH              SIZE <string>
```

A numeric parameter is passed as:

```
PUSH              OFFSET <rtpnum structure>
```

The rtpnum structure is defined in the include file rtpall.inc.

To conform to pascal calling conventions, use the form of the "RET" instruction that automatically pops parameters from the stack (4 bytes per numeric parameter + 8 bytes per string parameter).

11.9.4 Linkage of Test Program

Programs written in Macro Assembler must be linked to produce an executable file. All input files to "ld" must be the result of previously run assemblies or libraries of files.

HINT: Although "ld" can be used to invoke the linker directly, it is easier to invoke it by using the "cc" command. The advantage of this is that "cc" automatically links in the C startup and support libraries which are required so they do not have to be named explicitly, provided the appropriately named libraries can be found in the standard system library directory (usr/lib).

On some systems, the linker may generate a "too many public symbols" warning. This is caused by the large size of NPL, and can be safely ignored.

The files required for production of the standalone should include:

- The mainline (e.g. mymain.o)
- The RTP test subroutine (e.g. myrtp.o)
- The RTPEXT subroutine (optional but recommended)

- The GOSUB' subroutines (e.g. mysub.o)

Implied in the linkage are:

- The C startup module crt1.o
- The C ending module crtn.o

For example:

```
cc -o mymain mymain.o myrtp.o myrtpext.o mysub.o
```

11.9.5 Linkage of Customized rti or rtp

As stated in the previous section, "cc" can be used to invoke the linker and automatically link the required C startup routines and support libraries.

The files required for production of the customized rti or rtp should include:

- ../lib/start (from BESDK)
- The mainline
- The RTPEXT subroutine
- The GOSUB' subroutines
- ../lib/rtplib (from BESDK)
- ../lib/rtpblib (Non-Interpretive rtp only)
- ../lib/srclib (Interpretive rti only)
- ../lib/srcblib (Interpretive rti only)
- ../lib/last (from BESDK)

Implied in the linkage are:

- The C startup module crt1.o

- The C ending module crtn.o

Linkage of .o object files produces by masm can be done using exactly the same commands as for object files produced by the C compiler. See the section on linking of customized rti or rtp with external subroutines written in C.

11.10 Flow Control for External Libraries

The flow control (in chronological order) for rtp or rti using external subroutines is as follows:

1. C startup routines execute and perform some initialization, and eventually call the external library mainline (i.e., main()).

2. The mainline performs some initialization work for the external subroutines, and calls RTP(). If assembly language is used, initialization of the memory allocation support module is required prior to calling RTP.

3. NPL runs and does some initial configuration work, including processing command-line options and loading the bootstrap program.

4. NPL scans the external library for numbered DEFFN's with named aliases, using the LIST' calls starting at function number 0. An internal table of identifiers and equivalent externals is built.

5. NPL execution proceeds. At some point, a GOSUB', (e.g., GOSUB'100) is executed, and no local GOSUB' subroutine is found. If the GOSUB' is to a named DEFFN', and the identifier is found in the table created in step 4, the equivalent number is used to query RTPEXT.

6. NPL calls RTPEXT to find out whether an external '100 subroutine exists, and if so, where it is and what parameter types it needs.

7. RTPEXT supplies the requested information (i.e., GOSUB'100 exists, has 3 parameters with types string, string, and numeric, which is located at mysub()) and returns (to NPL).

8. If the RTPEXT indicated that the subroutine does not exist, or if the number and type of parameters do not match, a NPL error is generated on the GOSUB' statement. Otherwise, NPL evaluates parameters and calls the external subroutine (mysub) whose address was provided by RTPEXT.

9. The external subroutine (mysub) executes and returns to NPL. NPL execution proceeds until we are back at step 5 (another GOSUB') or the rtp is ending ($END, Killed from HELP, etc.). In the second case, go to the next step.

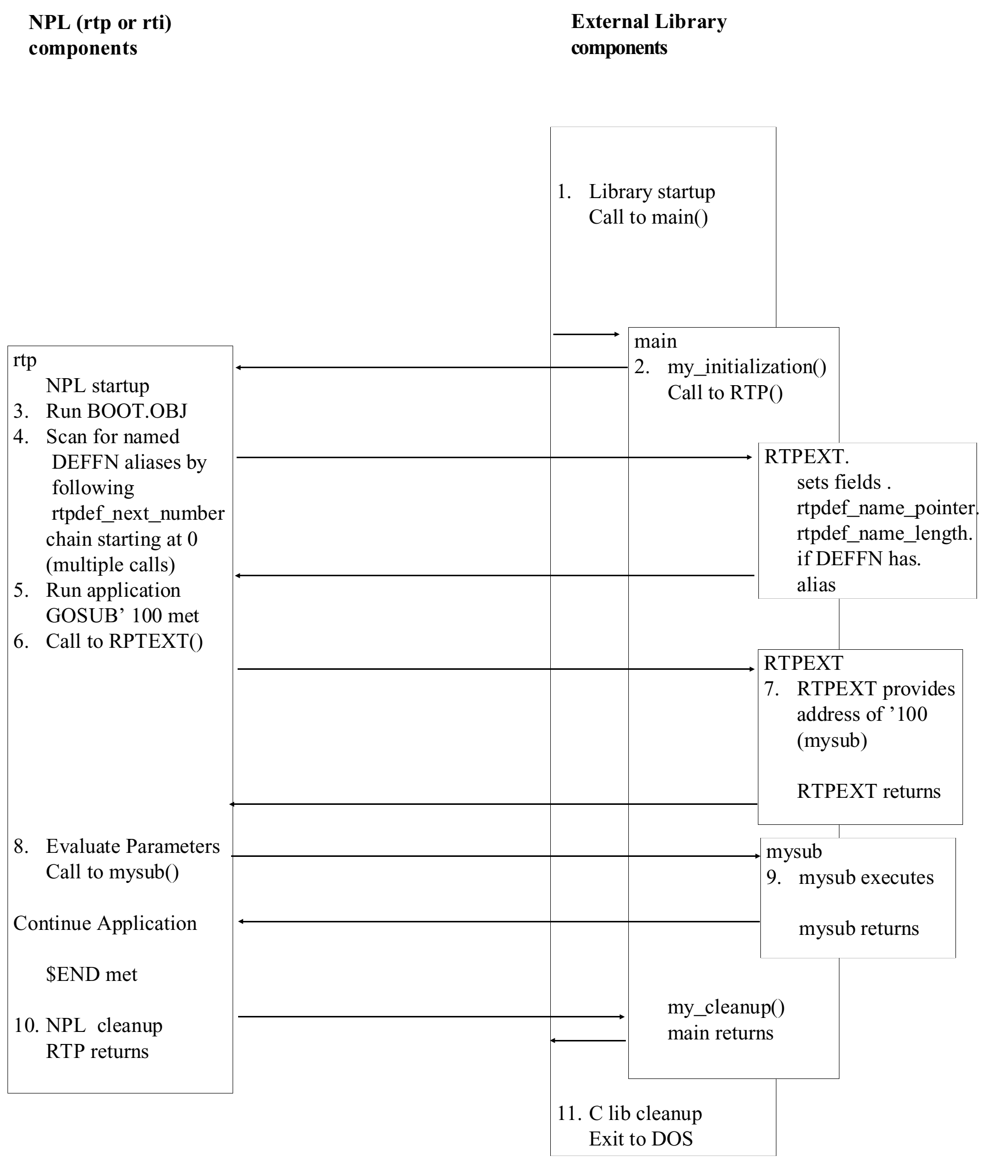
10. NPL does its cleanup, then the RTP () subroutine returns to the caller (in the external library mainline).

11. The external library mainline does C or Pascal library shutdown, and eventually exits back to UNIX.

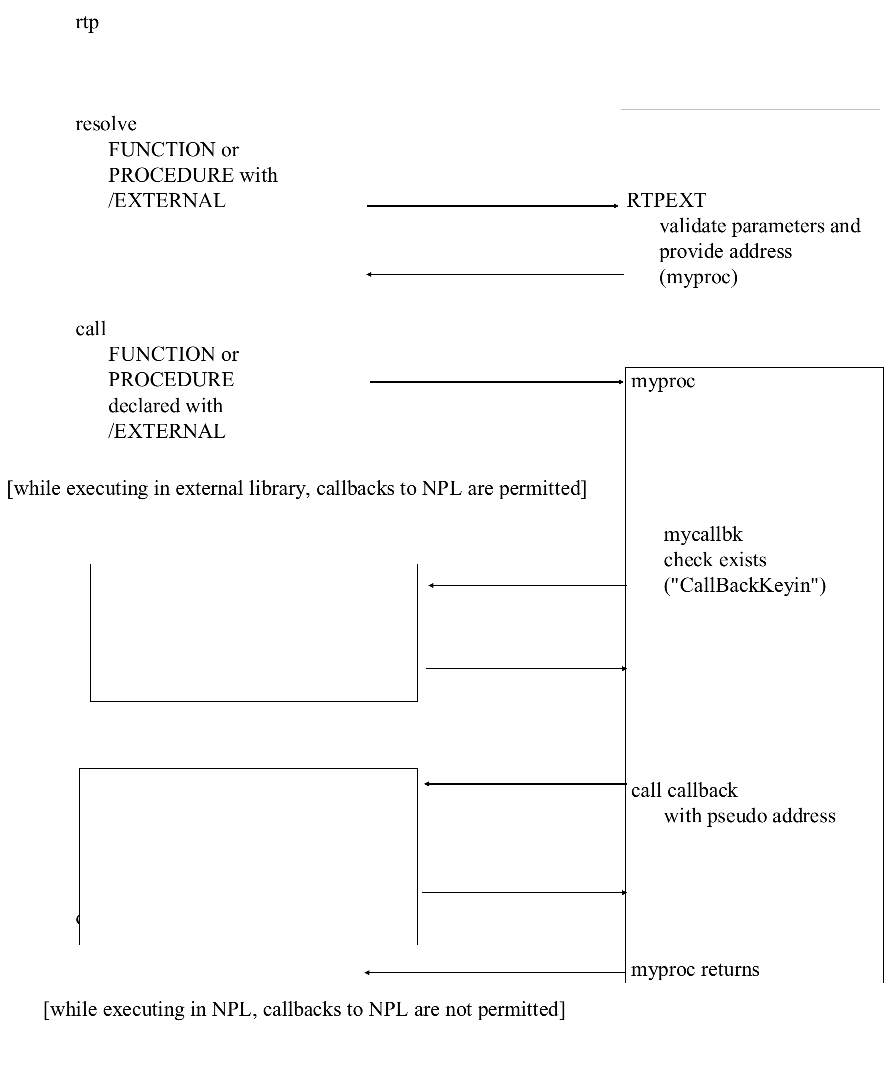
NOTE: RTPEXT can be called by LIST' to find out information about DEFFN' subroutines without actually calling the subroutines.

Subroutines should not DEPEND on the above flow control order to work (i.e., a subroutine should not expect RTPEXT to always be called immediately before it).

The above outline is provided merely as a guide to your understanding of how the external mainline, rtp, RTPEXT and external subroutines interact.he following diagram shows how execution proceeds using the various software components with the above steps labeled.


NPL (rtp or rti) components
External Library components
1. Library startup
Call to main()
main
2. my_initialization()
Call to RTP()
rtp
NPL startup
3. Run BOOT.OBJ
4. Scan for named DEFFN aliases by following rtpdef_next_number chain starting at 0 (multiple calls)
5. Run application GOSUB' 100 met
6. Call to RPTEXT()
RTPEXT.
sets fields .
rtpdef_name_pointer.
rtpdef_name_length.
if DEFFN has.
alias
RTPEXT
7. RTPEXT provides address of '100 (mysub)
RTPEXT returns
8. Evaluate Parameters
Call to mysub()
mysub
9. mysub executes
mysub returns
Continue Application
$END met
10. NPL cleanup
RTP returns
my_cleanup()
main returns
11. C lib cleanup
Exit to DOS

The following diagram shows the execution picture for the FUNCTION/PROCEDURE (interface):


rtp
resolve FUNCTION or PROCEDURE with /EXTERNAL
RTPEXT validate parameters and provide address (myproc)
call FUNCTION or PROCEDURE declared with /EXTERNAL
myproc
[while executing in external library, callbacks to NPL are permitted]
mycallbk check exists ("CallBackKeyin")
call callback with pseudo address
myproc returns
[while executing in NPL, callbacks to NPL are not permitted]

APPENDIX A

COMMON PROBLEMS

A.1 Overview

Many of the problems discussed below are associated with how files and devices are accessed under the UNIX operating system. These problems typically fall into these categories:

1. Locating files and programs. If directories are specified explicitly in boot programs or in compiler batch files, be sure to include the /usr specification. Also be sure that the alternate path is properly set up for the user from the .profile file or set the NIAKWA_RUNTIME environment variable is not set. Refer to Section 2.3.2 for details on the .profile file.

2. Access privileges to files, devices, and programs. If attempting to access a given file or device, make sure that the user has read and/or write privileges to the file or device. Read and/or write privileges must also exist for the directory in which the file or device is located. For programs (rti, rtp, b2c, and shell script files), the user must have execute privileges. Refer to the UNIX Commands Directory for details on establishing or changing access privileges.

3. Case sensitivity under UNIX. UNIX filenames are case sensitive. If a P48 error occurs while executing the NPL RunTime check the $DEVICE statements for proper case of the file names.

Section A.2 discusses problems and errors generated by the RunTime.

Section A.3 discusses the general problems and errors generated by the Niakwa NPL Upgrade RunTime program.

Section A.4 discusses problems and errors generated by the NPL Compiler.

A.2 Runtime Problems

This section describes specific problems that may be encountered using the NPL RunTime program along with possible solutions.

Problem 1: Program rtp (or rti) not found.

The rtp or rti programs are not in the /usr/BASIC2C directory.

The rtp or rti programs do not have execute privileges.

The /usr/BASIC2C directory has not been properly set up as an alternate path.

The directory specified by the NIAKWA_RUNTIME environment variable has not been properly set up as an alternate path.

Install the NPL RunTime Package.

Use the UNIX "chmod" command to set up execute privileges for /usr/BASIC2C/rti or /usr/BASIC2C/rtp (run setup.nodes).

Modify the user's .profile file as described in Section 2.3.2. It is necessary for the user to logoff and login after this change has been made in order for it to take effect.

Set the NIAKWA_RUNTIME environment variable to point to the correct directory or modify the user's .profile as described in Section 2.3.2.

NOTE: This problem may also apply to any shell script files used to invoke the NPL Run-Time. The same solutions apply.

Problem 2: A RunTime error P48 is generated.

The NPL RunTime program was unable to open the specified diskimage file.

The user does not have access rights to the specified diskimage file.

The $DEVICE statement used for raw diskette access may be wrong.

Be sure that the $DEVICE equivalence is specified correctly (case sensitive). If the $DE-VICE statement does not include a directory specification (e.g., is simply a file name - PLATTER1.BS2), be sure that the current directory is established properly with a cd statement before execution of the RunTime.

Use the UNIX "chmod" statement to modify access rights. Read/write access is required for the diskimage. The directories in which the diskimage file resides must also be set up with appropriate read/write privileges.

If the device being accessed is a 5-1/4" 360K, or 1.2MB or 3-1/2" 720K, 1.44MB, or 2.88MB "raw" diskette, check that the $DEVICE statement references the proper UNIX device. Refer to Section 5.2 for the proper $DEVICE statement.

Be sure that these devices have been created in the /dev directory and that the user has read/write privileges for them. Refer to Section 5.2 for details.

Problem 3: The NPL RunTime program generates an I90 (or any other I series) error.

The user may not have read/write privileges to the diskimage file or "raw" device being accessed.

Use the chmod command to set up proper access privileges.

Problem 4: The NPL RunTime program generates an immediate D82 error stating that the boot program cannot be found.

The current directory has been improperly defined.

The boot program exists, but the case is different than specified.

The boot program exists, but the .OBJ extension is not in uppercase.

Check the shell script files used to invoke the NPL RunTime.

Either modify your shell script file or rename the boot program.

The NPL RunTime program will only look for boot programs with an uppercase .OBJ extension.

Problem 5: When reading from "raw" diskettes, data from a diskette previously in the drive is accessed.

This problem is due to the restrictions that UNIX imposes on accessing diskettes. Whenever the diskette drive door is opened, UNIX requires that the logical file associated with the diskette drive be closed and reopened.

NPL programs which access diskettes must use either $CLOSE statements or a "$DEVICE (#X)=$DEVICE(#X)" statement (where X is the NPL device number used for the diskette) whenever a new diskette is mounted (before the first access to the new diskette) and after the last access to the last diskette (before exiting the program). Refer to Section 5.2 for further details on diskette handling under UNIX.

Problem 6: "Cannot Determine Terminal Number" is displayed upon execution of rti or rtp.

The /usr/BASIC2C/ttys file is not properly setup or does not exist.

The system is in single user mode.

Create or properly setup the /usr/BASIC2C/ttys file. Refer to Section 7.4 or 8.3 for more details.

Bring system up in multi-user mode.

Problem 7: The 2200 character set does not display properly on the screen.

The rtp or rti is unable to load the proper NPL screen translation file.

The SCREEN translation file does not exist. Reinstall from the Terminal Support files diskette or recopy to the end user's system from your system.

The user does not have read privileges to the file. Use the chmod command to correct this.

The characters may not be displayable on the terminal in use, or for terminals that support downloadable fonts, the appropriate font file may not have been downloaded to the terminal.

The NPL RunTime may think that it is working with a different terminal than what is in use. Check the UNIX system variables TERM or BASIC2C_TERM to verify that the correct terminal type is set.

Problem 8: The keyboard does not generate expected key codes.

The rtp or rti is not able to load the NPL keyboard translation file

The KEYBOARD translation file does not exist.

The user does not have read privileges to the file.

The characters may not be defined on the keyboard in use.

The NPL RunTime may think that it is working with a different terminal than what is in use.

Reinstall from the Development Package diskette.

Use the chmod command to correct this.

Download the keyboard font file if applicable.

Check the UNIX system variables TERM or BASIC2C_TERM to verify that the correct terminal type is set.

Problem 9: The message "the Niakwa Programming Language authorized user limit exceeded. Access denied." appears.

The maximum number of authorized users for the NPL RunTime Package has been exceeded.

Wait until one of the current users exits the NPL RunTime program. Access is then allowed.

The problem may have also been caused by a situation where the user count is greater than the actual number of NPL users on the system. Typically this occurs if tasks are killed with a kill -9 command. The /usr/BASIC2C/userfix utility allows the developer to reset the user count in this situation (UNIX only, Xenix users must reboot or rename the commonmem file in /usr/BASIC2C). Refer to Section 7.6.1.

Problem 10: The message "Interpreter not enabled." appears.

The Interpretive RunTime Package (rti) has not been enabled on the system.

Use the Non-Interpretive version of the NPL RunTime program (rtp), or install the ENABLED file (found on the NPL Development Package Compiler Diskette) in the /usr/BASIC2C directory.

Problem 11: The message "cannot execute Access denied" appears.

The rti or rtp program does not have execute privileges.

Ensure that the rtp and rti file has execute privileges. The /usr/BASIC2C/setup.nodes shell script will ensure that this is done.

Problem 12: The message "No more process available" appears. In addition, the message "no files" may appear on the console.

The default UNIX files and inodes parameters are inadequate for use with NPL. This can be cause by a high number of terminals being used with NPL, or a lower number of terminals is being used, but each terminal is using a large number of files.

To correct this situation, it is necessary execute the UNIX Configure utility (typically /usr/sys/conf/configure). Niakwa suggests that the following parameters be modified:

Option: Files, Inodes, and Filesystems			
Parameter	Description	**From**	**To**
(NINODE)	Inodes	100	150
(NFILE)	Open Files	100	150
Option: Processes, Memory, and Swapping			
Parameter	Description	**From**	**To**
(NPROC)	Max Processes	60	100

NOTE: The above setting should be adequate for a sixteen user system. Higher values may be required for a higher number of terminals or other special cases. Refer to the UNIX User's Guide for more information on using the UNIX "configure" utility.

Problem 13: The message: "Serial number not found on floppy diskette. Please contact your distributor for assistance the NPL RunTime Canceled" appears.

The GOLD KEY diskette in the drive is not the Gold Key that is installed on the system.

Use the original Gold Key diskette that was used for the installation.

Verify that the drive door is closed.

Problem 14: The message "Please mount your Gold Key diskette" appears on the screen when executing the NPL RunTime.

The Gold Key diskette is not mounted in the floppy drive.

Place the Gold Key diskette in the floppy drive and continue executing the NPL RunTime.

Check to see that the NPL RunTime has been installed. This can be done by running the "xeninst" installation program. If the installation was not done, the installation program continues. Otherwise, the installation program aborts with the error message "No installs left on this diskette."

Problem 15: When executing the Gold Key installation program, the message "Enter 'xeninst I' to install or 'xeninst D' to recall NPL."

An attempt was made to install or recall security without specifying the appropriate parameters.

Reenter the xeninst command making sure that the correct program is specified. For example:

```
./xeninst X Y
```

where: X = I for install or D for de-install.

Y = 0 for drive 0 or 1 for drive 1 (drive 0 is the default).

Problem 16: The message "No installs left on this diskette" appears when trying to install the security.

The security is already installed on another system.

Recall the protection from the other system.

Problem 17: The message "Serial number not found on floppy diskette" appears when trying to recall the security.

The original Gold Key diskette is not in the floppy drive.

Verify that the original Gold Key diskette is in the floppy drive.

Problem 18: The message "NIAKSER.DAT missing or damaged (Error Code) appears on the screen when executing the NPL RunTime.

The NIAKSER.DAT file is not in the /usr/BASIC2C directory or the directory specified by the NIAKWA_RUNTIME environment variable.

Invalid data is contained in the NIAKER.DAT file.

Make sure that the NIAKSER.DAT file is in the /usr/BASIC2C directory. It may be necessary to copy this file again from the original Gold Key diskette.

Below are the description for the Error Code) values that may occur.

Error Code	**Description**
Code 0	Cannot find NIAKSER.DAT.
Code 1	Cannot read NIAKSER.DAT.
Code 2-5	Invalid data in NIAKSER.DAT. File is probably damaged

Problem 19: The message "Please Mount your Gold Key diskette" remains on the screen with an error code 1706 displayed on the screen after the NPL RunTime has been executed.

The encrypted serial number in the NPL RunTime on the hard drive does not match the encrypted serial number on the Gold Key diskette. This would occur when attempting to use a different diskette than was used to install the NPL RunTime.

Be sure that the Gold Key that is mounted is the original diskette that was used to install the NPL RunTime.

Problem 20: An error code "1706" appears in the upper left hand corner during the security install process or when starting up the NPL RunTime.

Security has not yet been installed on the hard drive. This message appears during the normal security install process.

Install the Gold Key security on the hard drive.

A.3 Upgrade Problems

This section describes general problems which may be encountered in upgrading an existing NPL Release III RunTime to Release IV with a Niakwa NPL Upgrade Package.

Problem 1: When upgrading, a "This upgrade is for a site with x" message appears.

The upgrade being used does not match the original system being upgraded. Contact Niakwa on obtaining the correct Niakwa NPL Upgrade Package for the system to be upgraded.

Problem 2: When upgrading, an Error #XX message appears.

Causes for this vary. The text for these various upgrade error messages are intended to be self explanatory. Correct the indicated problem or contact Niakwa for information on obtaining the correct Niakwa NPL Upgrade Package for the system to be upgraded.

A.4 Compiler Problems

This section describes specific problems and their solution that may be encountered when using the NPL compiler.

Problem 1: Program b2c not found.

The b2c program is not in the /usr/BASIC2C directory. Install the compiler.

The user does not have execute privileges for the file b2c or the directory /usr/BASIC2C.

The /usr/BASIC2C directory has not been properly set up as an alternate path.

Modify the user’s .profile file as described in Section 2.3.2. It is necessary for the user to logoff and login after this change has been made for it to take effect.

Use the chmod command to set up execute privileges. Refer to the UNIX Commands Directory for details on the chmod command.

NOTE: **This problem may also apply to any shell script files used to invoke the compiler. The same solutions apply.**

Problem 2: The compiler generates a message "No source programs matching wildcard".

The compiler could not find any programs in the "srcloc" specified which matched any of the program names or program name wildcards entered.

The "srcloc" has been improperly specified.

If attempting to compile ASCII text programs, be sure that the .SRC extension is in uppercase as the file is named. The compiler ignores programs with lowercase .src extensions.

If using a batch file to compile programs, be sure that the "srcloc" includes /usr as part of the path name.

Problem 3: The compiler states "cannot open "srcloc" (or "objloc") as a diskimage file" when compiling from or to diskimage files.

The compiler cannot locate the specified diskimage file.

The user does not have access rights to the specified diskimage file.

Be sure that the "srcloc" (or "objloc") is specified correctly (case sensitive). If the "srcloc" (or "objloc") does not include a directory specification (i.e., is simply a file name - PLATTER1.BS2 for example), be sure the current directory has been properly specified.

Use the UNIX "chmod" statement to modify access rights. Read access is required for the "srcloc" diskimage. Read/write access is required for the "objloc" diskimage. The directories in which the diskimage file resides must also be setup with appropriate read/write privileges.

Problem 4: The compiler states "cannot open "srcloc" as a diskimage file" when compiling from 2200 format diskettes.

The "srcloc" is not properly specified.

The "srcloc" does not have read access privileges.

NOTE: The Wang 2200 does not directly support 3-1/2" diskettes.

Be sure that the "srcloc" is properly specified. For compiling from 2200 format diskettes, the specification is:

```
/dev/fd048ds9
```

Be sure that the user has read access privileges to the device (in the /dev directory). Use the chmod command to change access to the device.

APPENDIX B

UNIX ERROR CODES

B.1 Overview

In the Intel UNIX implementation of the Niakwa RunTime program, the Error Processor displays a UNIX Error Code (displayed on the screen as "UNIX Error Code: xxxx") whenever possible. This code shows the operating system error code received by the NPL RunTime during execution of the statement producing the error. This code contains information which is useful in diagnosing the problem. Refer to the appropriate UNIX reference manual for further details on the error message.

All UNIX Error Codes are reported in hexadecimal format, not decimal. The following are some of the more frequently encountered UNIX error codes and their meanings:

0002 File not found. This code is most often displayed with a NPL error code P48 when an attempt is made to access a diskimage file or UNIX device which does not exist. This error may also be displayed with an initial D82 error when the specified BOOT program is not found.

0005 General I/O error. This code is most often displayed with a NPL error code of I93. It usually indicates a disk format error. This code is also returned when an attempt is made to access a diskette that is not in the drive.

000D Permission Denied. This error code indicates that the user does not have proper access privileges for the file or device being accessed. It is most often associated with a NPL error code of I90.

NOTE: Located on the NPL RunTime diskette are two files, RTIXERR.HLP and RTIXERR.IDX, which must be installed in the /usr/BASIC2C directory (or the specified default RunTime directory) to display descriptive error messages for all UNIX operating system error codes.

APPENDIX C

PERFORMANCE ISSUES

C.1 Overview

The purpose of this Appendix is to provide suggestions on ways to manage performance when using UNIX-based operating environments with NPL.

Section C.2 discusses the use of RAM disks.

Section C.3 discusses $OPTIONS settings.

Section C.4 discusses configuration issues.

C.2 RAM Disks

UNIX operating environments typically support the use of RAM disks. Performance can be significantly enhanced by implementing this feature, especially in the areas of program loading and record locking. Refer to Section 5.3.4 and the UNIX documentation for details.

C.3 $OPTIONS Settings

There are several $OPTIONS setting that can have a significant impact on NPL performance under UNIX. These include:

- Byte 14 for timeslice release. Refer to Section 5.3.3, 7.5.2, 8.2.2, and the NPL Statements Guide, $OPTIONS.
- Byte 35 for no $BREAK behavior. Refer to Section 7.5.2, 8.2.2 and the NPL Statements Guide, $OPTIONS.
- Byte 39 for no implicit $OPEN. Refer to Section 5.3.6, 7.2.1, 8.2.2, and the NPL Statements Guide, $OPTIONS.

C.4 Configuration Issues

There are several NPL performance issues under UNIX that can be considered related to configuration. These include:

- The use of polling KEYINs and $BREAK. Refer to Section 5.3.3 for options related to this UNIX specific issue.
- Using read only files to avoid file locks. Refer to Section 7.2 for details.

APPENDIX D

"RAW" DEVICE COMPATIBILITY CHART

D.1 "Raw" Diskette Chart

This chart lists the "raw" diskette compatibility for all operating system groups currently supported by NPL.

NPL "RAW" DEVICE COMPATIBILITY TABLE									
				5-1/4" Diskettes			3-1/2" Diskettes		
System	Drive Type	$FORMAT	320K	360K	720K	1.2MB	720K [8]	1.44MB [8]	2.88MB[9]
Wang 2200/CS	360K	X	X	X[1]	-	-	-	-	-
	1.2MB*	X[2]	-	-	-	-[3]	-	-	-
MS-DOS	360K	X	X	X[4]	-	-	-	-	-
	720K	X	-	-	-	-	X[4]	-	-
	1.2MB*	X	X	X[4]	-	X[4]	-	-	-
	1.44MB	X	-	-	-	-	X[4]	X[4]	-
	2.88MB	X	-	-	-	-	X[4]	X[4]	X[4]
Intel UNIX &Xenix	360K	[5]	-	X	X	-	-	-	-
	720K	[5]	-	-	-	-	X	-	-
	1.2MB*	[5]	-	X	X	X	-	-	-
	1.44MB	[5]	-	-	-	-	X	X	-
	2.88MB	[5]	-	-	-	-	X	X	X
SuperDOS	360K	X[6]	-	X[4]	-	-	-	-	-
	720K	X[6]	-	-	-	-	X[4]	-	-
	1.2MB*	X[6]	-	X[4]	-	X[4]	-	-	-
	1.44MB	X[6]	-	-	-	-	X[4]	X[4]	-
	2.88MB	-	-	-	-	-	-	-	-
Honeywell XPS-100	720K	[7]	-	-[7]	X	-	-	-	-
Bull DPX/2	720K	[7]	-	-[7]	X	-	-	-	-
NCR TOWER	1.2MB	[10]	X[11]	X[11]	X[11]	-	-	-	-
IBM RS/6000	1.44MB	[5]	-	-	-	-	X	X	-

X = SUPPORTED

* Any 360K diskette (DSDD) which has been written to in a 1.2MB drive may be unreliable when accessed on a 360K drive. This is a stated hardware limitation of the 1.2MB drive technology.

NOTE: Diskettes which have been written to on 1.2MB drives can always be read successfully on another 1.2MB drive. In addition, diskettes which have been created on a 360K drive can always be read successfully on any 1.2MB drive. This restriction applies only to reading diskettes on a 360K drive which have been written to on a 1.2MB drive.

NOTES:

(1) Using the "PC Interchange" format, a special $GIO microcommand sequence is required to format 360K diskettes. Refer to Section 15.7.1 of the Programmer's Guide for details.

(2) Neither 360K nor 320K diskettes can be formatted in the 1.2MB diskette drive on the DS. 1.2MB diskettes created in the native 2200 format (256 byte sectors) on the Wang DS are not supported on any NPL machine.

(3) Although the Wang documentation specifies that "PC Interchange" format is supported on the 1.2MB diskette, our testing to date has not yielded positive results.

(4) Access to 360K, 1.2MB, 720K, 1.4MB, and 2.88MB "raw" diskettes under MS-DOS and SuperDOS (excluding 2.88MB) is supported by RTP and RTI, but not by B2C, which requires 320K "raw" formatted diskettes. Under Release IV, Super-DOS, 320K "raw" formatted diskettes are not supported under B2C.

(5) $FORMAT DISK is not supported under Intel UNIX or Xenix. Refer to the Intel UNIX Supplement for details.

(6) $FORMAT DISK for 3-1/2" or 5-1/4" inch diskettes is not supported under Protected Mode SuperDOS.

(7) Many restrictions apply to the use of 360K "raw" diskettes on the Honeywell Bull XPS-100 and Bull DPX/2. Refer to Section 10.3 of the appropriate UNIX Addendum in the NPL Release III UNIX V Supplement for more information.

(8) Support of 3.5" 720K and 1.44MB diskettes requires Release 3.00 or higher.

(9) Support of 3.5" 2.88MB diskettes requires Release 4.00 or higher.

(10) The $FORMAT DISK command is not supported for any "raw" diskettes on the NCR Tower 32.

(11) Only read and write operations of "raw" diskettes are supported. It is possible to format 640K diskettes using the UNIX format command. According to the NCR documentation, 720K diskettes are not supported. This is because the UNIX format command does not always succeed when a 720K diskette is specified. As a result, you may not be able to format 720K diskettes on the NCR TOWER 32 systems. It is not possible to format 320K or 360K diskettes. Therefore, 320K and 360K diskettes must be preformatted on another system.

For information regarding naming conventions and accessing "raw" diskettes within specific hardware environments, refer to Chapter 5 of the appropriate NPL Supplement.