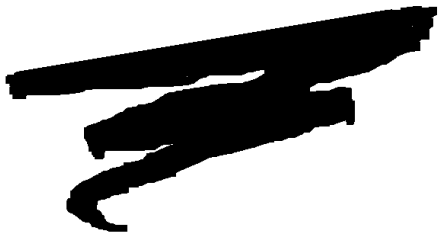
NIAKWA PROGRAMMING LANGUAGE

MS-WINDOWS ADDENDUM



1st Edition - August 1993

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

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

DISCLAIMER OF WARRANTIES AND LIMITATION OF LIABILITIES AND PROPRIETARY RIGHTS

The staff of Niakwa, Inc. (Niakwa) has taken due care in preparing this manual. Nothing contained herein shall be construed to modify or alter in any way the standard terms and conditions of the Niakwa Programming Language (NPL) Support and Distribution License Agreement, the End-User Support Only License Agreement, the Niakwa Software License Agreement and Warranty and any other Niakwa License Agreement (collectively, the "License Agreements") by which this software package was acquired.

This manual is to serve as a guide for use of the Niakwa software only and not as a source of representations or additional undertakings by Niakwa. The licensee must refer to the License Agreements for Niakwa product and service representations.

No ownership of Niakwa software is transferred by any of the License Agreements. Any use of Niakwa software beyond the terms and conditions of the License Agreements, without the written authorization of Niakwa, is prohibited.

PREFACE

This Niakwa Programming Language (NPL) Addendum for MS-Windows is designed as an addition to the NPL Supplement for MS-DOS. This Addendum discusses the installation, operation, and MS-Windows specific features of the Niakwa Interpreter and Run-Time Program. For more information, refer to the appropriate NPL documentation and the MS-Windows documentation.

1.1 Prerequisite Knowledge

This Addendum assumes at least a basic knowledge of the IBM Personal Computer, the Microsoft Disk Operating System (DOS) Version 3.10 or greater, and MS-Windows Version 3.1 or greater.

This Addendum also assumes an understanding of the information contained in the NPL MS-DOS Supplement.

1.2 How to Use this Addendum

This addendum should be used by the developer as a guide to understand how to create and modify applications for use with NPL for MS-Windows.

All chapters should be reviewed thoroughly by the developer. Below is a summary of the topics discussed in each chapter.

Chapter 1 introduces the MS-Windows RunTime, discusses the NPL Development and RunTime Package diskette contents, and the specific features of the NPL RunTime under MS-Windows.

Chapter 2 discusses the installation procedures necessary for NPL under MS-Windows.

Chapter 3 discusses the exact configuration requirements for NPL under MS-Windows.

Chapter 4 discusses the startup, operation, and closing of RunTime tasks under MS-Windows.

Chapter 5 discusses the differences in device support for NPL under MS-Windows.

Chapter 6 discusses the multi-user capabilities made available under MS-Windows.

Chapter 7 discusses the operating environment-specific language statements under MS-Windows.

Chapter 8 discusses the use of the External Call feature of NPL under MS-Windows.

Appendix A provides information on the common problems that may occur using the MS-Windows RunTime.

Appendix B provides information on modifying the Niakwa Fonts.

Appendix C provides an example of DDL use.

Appendix D provides information on MS-Windows performance issues.

NOTE: This Addendum is intended to cover environment-specific differences from the generic NPL information provided in the NPL Programmer's Guide and Statements Guide or the operating system-specific information contained in the NPL MS-DOS Supplement.

TABLE OF CONTENTS

PREFACE

INTRODUCTION

INSTALLATION

CONFIGURATION

RUNTIME OPERATION

DEVICE SUPPORT

MULTI-USER CAPABILITIES

PLATFORM-SPECIFIC LANGUAGE FEATURES

MIXED LANGUAGE PROGRAMMING

COMMON PROBLEMS

MODIFYING NPL FONTS

EXAMPLE DYNAMIC LINK LIBRARIES

PERFORMANCE ISSUES

CHAPTER 1

INTRODUCTION

1.1 Overview

The NPL MS-Windows Addendum is intended as an aid in the correct installation and use of the MS-Windows version of the Niakwa Development Package and RunTime programs.

NOTE: This Addendum details the additional features of NPL operating under MS-Windows. Refer to the MS-DOS Supplement for information on the standard NPL features.

Section 1.2 describes the contents of the NPL MS-Windows Supplementary Files Diskette.

Section 1.3 describes the additional contents of the NPL RunTime Package for MS-Windows.

Section 1.4 discusses the MS-Windows specific features of the NPL RunTime program for MS-Windows.

1.2 Contents of MS-Windows Development Package

The NPL Development Package is intended for software developer use in the development and execution of application software on MS-DOS or Novell NetWare systems using MS-Windows. The Development Package for MS-Windows is the same as for MS-DOS with the addition of the MS-Windows Supplementary Files Diskette.

The contents of the standard MS-DOS Development Package diskettes are listed in Section 1.2 of the MS-DOS Supplement.The following is a description of the additional MS-Windows-specific files included on the MS-Windows Supplementary Files Diskette.

\BFONTS	All files necessary for modification of the Niakwa font file BASFONTS.FON with the MS-Windows Software Development Kit or other third-party font creation products. Refer to Appendix B for complete information.
\IFONTS	All files necessary for modification of the Niakwa font file IBMFONTS.FON with the MS-Windows Software Development Kit or other third-party font creation products. Refer to Appendix B for complete information.
\WINCDEMO	This directory contains the example DLL files which illustrate how NPL can be used to interface with MS-Windows resources. Refer to Appendix C for detailed information.
\WINCDIAL	This directory contains example DLLs and other related files which illustrate how NPL can be used to control a Window Dialog box. Refer to Chapter 8 and Appendix C for details.
\	All other files contained in other directories on this diskette are for the MS-Windows version of the NPL external call interface to programs written in other languages. Refer to Chapter 8 for more information.

1.3 Contents of MS-Windows RunTime Package

NOTE: The MS-Windows RunTime is designed to work only in the presence of a special version of the Niakwa RunTime security.

The MS-Windows RunTime Package physically consists of the Niakwa RunTime Package User's Guide and two 5-1/4" or three 3-1/2" diskettes.

The contents of the standard RunTime files are listed in Section 1.3 of the MS-DOS Supplement. The following is a description of the additional MS-Windows specific files included on the MS-Windows RunTime disk(s).

MS-Windows RunTime Package Files

BASFONTS.FON	The Niakwa screen fonts provided for use with MS-Windows for the standard NPL character set.
IBMFONTS.FON	The Niakwa screen fonts provided for use with MS-Windows for the standard IBM character set.
LAUNCH.EXE	This file is used with the INVOKE ($SHELL) command to allow the MS-Windows RunTime to start another task without waiting for the task to complete.
RTINERR.HLP	This file is an text file that contains the MS-Windows error messages that are displayed optionally when using the Interpretive RunTime program for MS-Windows.
RTINERR.IDX	This file contains the index listings used when the RTINERR.HLP file is accessed.
RTIWIN.EXE	The NPL Interpretive MS-Windows RunTime program.
RTPWIN.EXE	The NPL Non-interpretive MS-Windows RunTime program.
SHAREDLL.DLL	A file used internally by the MS-Windows RunTime.

1.4 MS-Windows Specific Features

The MS-Windows version of NPL contains the features documented in Chapter 1 of the MS-DOS Supplement and the following additional features:

- Support for the standard NPL RunTime so that applications that are not MS-Windows compatible can still be run with the MS-Windows RunTime Package.

NOTE: This allows for any combination of MS-Windows and standard RunTime users up to the available user limit on network installations.

- Multi-tasking with true multi-user capabilities including unique task identification.
- The user limit is only decreased for the first RunTime task started under MS-Windows (subsequent MS-Windows RunTime tasks do not affect the user limit).
- Extended memory support beyond 640K. As much memory as is physically available to MS-Windows is available to applications written with NPL.
- Operates in MS-Windows Standard or 386 Enhanced modes.
- Optional automatic resizing of fonts. Niakwa has provided a series of fonts specifically designed for use with NPL as well as instructions to modify these and create new font files.
- Mouse support directly through the NPL MS-Windows RunTime or through the Browse and SF Keys interactive option.
- 132-column screen support.
- Use of Dynamic Link Libraries (DLL) is supported for use with external calls. This feature allows the many MS-Windows specific features to be used (dialog boxes, text displays, message boxes, clipboard interface, etc.). NPL allows "C" routines in DLLs to call NPL FUNCTIONs or PROCEDUREs. NPL also provides limited support for MS-Windows messages. Example DLLs with source code that shows many of these features are provided in Appendix C of this Addendum.

- Fixed pitch MS-Windows True Type Font support
- $SHELL can either immediately return control back to the RunTime or respond as it does under the standard NPL RunTime for MS-DOS.

CHAPTER 2

INSTALLATION

2.1 Overview

This chapter provides instructions for installing the Niakwa Development and RunTime Packages for MS-Windows (where differences exist from those documented in Chapter 2 of the NPL MS-DOS Supplement).

Section 2.2 discusses the operating system, memory, and hardware requirements for the NPL Software under MS-Windows.

Section 2.3 discusses the configuration requirements for NPL.

Section 2.4 discusses the installation of NPL MS-Windows Development software.

Section 2.5 discusses NPL Gold Key security for MS-Windows.

Section 2.6 discusses the installation of the NPL RunTime Package.

Section 2.7 discusses the installation of the NPL Upgrade RunTime.

2.2 MS-Windows and Hardware Requirements

The MS-Windows version of the NPL is designed to operate on systems meeting the following requirements:

- An IBM PC/AT or compatible (80286 or higher) operating under MS-DOS 3.10 or higher.
- MS-Windows 3.1 or higher.
- Meeting all MS-Windows 3.1 hardware requirements.
- Developers who wish to work with external calls or who need to modify the font files provided by Niakwa (refer to Appendix B), require the MS-Windows Software Development Kit (SDK) version 3.1 or higher.

2.3 NPL Configuration Requirements

This section discusses the specific configuration requirements for the MS-Windows RunTimes that differ from those documented in Chapter 2 of the NPL MS-DOS Supplement. Refer to Appendix D for additional configuration information on enhancing performance.

2.3.1 Files

The number of open files established in the CONFIG.SYS file should be changed to 45 or higher for MS-DOS based systems and 61 or higher for Novell NetWare based systems. This can be accomplished by editing the CONFIG.SYS file with a text editor (i.e., the MS-Windows Notepad, EDIT, EDLIN, etc.) and changing or adding the FILES parameter as shown below.

```
FILES=45 - for MS-DOS systems

FILES=61 - for Novell Netware systems
```

2.4 Installing the NPL Development Software

The following steps outline the installation of the MS-Windows Supplementary Files Diskette.

NOTE: The Niakwa NPL Development Package should be installed before the MS-Windows Supplementary Files Diskette is installed. Refer to Chapter 2 of the NPL MS-DOS Supplement for details.

2.4.1 Installing the MS-Windows Supplementary Files Diskette

There are no files on the MS-Windows Supplementary Files Diskette that are necessary for the NPL Development Software. Refer to Chapter 2 of the NPL MS-DOS Supplement for details on installing the standard NPL Development Software.

2.4.2 Installing the BESDK

Two versions of the BESDK (NPL, formerly Basic-2C, External Subroutine Development Kit) are provided with the Niakwa NPL Development Package. The first is contained on the diskette labeled MS-DOS BESDK Diskette and is for the standard Niakwa NPL RunTime. The second is the MS-Windows BESDK that is contained on the MS-Windows Supplementary Files Diskette. Refer to Chapter 11 of the MS-DOS Supplement for details on the standard BESDK and Chapter 8 of this Addendum for the MS-Windows-specific BESDK.

2.4.3 Installing the Example .DLL Files

The MS-Windows Supplementary Files Diskette also contains a series of example dynamic Link Library files illustrating how NPL can make use of the various MS-Windows resources. These examples and all associated files are automatically installed when the MS-Windows BESDK files are installed. Refer to Appendix C for a detailed description of the example DLL files and Chapter 8 for information on the MS-Windows-specific BESDK.

2.4.4 Installing the Font Source Files

The /BFONTS and /IFONTS directories on the MS-Windows Supplementary Files Diskette contain source files used to create the BASFONTS.FON and IBMFONTS.FON font files provided with the NPL RunTime. These may be modified by the developer. Refer to Appendix B for installation instructions and further information.

2.5 Niakwa RunTime Security for MS-Windows

Before installing the MS-Windows RunTime Package, an understanding of the MS-Windows RunTime security is required. This security is based on the proper installation of the standard Niakwa RunTime Package (refer to Chapter 2 of the MS-DOS Supplement for details).

2.5.1 NPL MS-Windows Gold Key Security

The MS-Windows Gold Key security operates slightly differently than that of the standard MS-DOS RunTime. These differences are discussed below. Refer to Chapter 2 of the MS-DOS Supplement for the procedure necessary to install the MS-Windows RunTime Gold Key Security.

2.5.2 The Gold Key Security TSR

Unlike the standard MS-DOS RunTime, the MS-Windows RunTime cannot automatically load the required security TSR (Terminate and Stay-Resident program). The proper TSR must be loaded before MS-Windows is started. It is recommended that this be done from the AUTOEXEC.BAT file or from a batch file used to start MS-Windows.

This can be accomplished by adding the following statement to the AUTOEXEC.BAT or batch file:

```
C:\BASIC2C\NIAKSECx
```

where x is one of the six files available (NIAKSECA.COM - NIAKSECF.COM). Typically, NIAKSECA is used. However, if a conflict occurs with an existing interrupt, attempt to use one of the other NIAKSECx files provided (NIAKSECB.COM - NIAKSECF.COM) until successful.

NOTE: The RTIWIN determines which interrupt to use based on the NiaksecVersion parameter (refer to Section 3.4 for details on the RTIWIN.INI file). If an interrupt other than NIAKSECA is used, an RTIWIN.INI file with the correct NiaksecVersion parameter setting must be created before the MS-Windows RunTime is executed.

2.5.3 User Limit

The user limit for the MS-Windows RunTime on Novell network installations is charged only for the first RunTime task started with the MS-Windows RunTime. All other RTPWIN windows opened (on the same workstation), after the initial window, have no effect on the user limit.

NOTE: Concurrent use of the MS-Windows version of the RunTime and the standard RunTime on the same system is not recommended. If this is done on a network workstation, both RunTimes count toward the user limit.

2.6 Installing the NPL RunTime Package

The installation of the MS-Windows RunTime is identical with that of the standard MS-DOS NPL RunTime except for the additional diskette(s), described in Section 1.3 of this Addendum. The additional diskettes should be installed in the same manner as described in Section 2.6 of the MS-DOS Supplement.

Once all diskettes have been copied to the hard drive, enter the following from the DOS prompt:

```
COPY C:\BASIC2C\*.FON C:\WINDOWS
```

This will copy the NPL font files for MS-Windows into the MS-Windows directory.

2.7 Installing the Upgrade RunTime Package

The Upgrade RunTime upgrades the existing NPL RunTime to Release IV. Refer to Section 2.7 of the NPL MS-DOS Supplement for details on installing the Upgrade RunTime Package.

NOTE: The upgrade procedures will overwrite the existing Release III RTIWIN.EXE and RTPWIN.EXE files. If it is necessary to go back the Release III versions of these files, they can be copied from the original Release III RunTime diskettes.

CHAPTER 3

CONFIGURATION

3.1 Overview

Once MS-Windows and the Niakwa Development and RunTime software has been installed, MS-Windows must be configured to work with the Niakwa software. This procedure is examined in this chapter.

Section 3.2 discusses configuring MS-Windows for use with NPL.

Section 3.3 discusses adding NPL RunTime tasks to the MS-Windows program manager.

Section 3.4 discusses the RTIWIN.INI file setup, organization, editing, and parameters.

Section 3.5 discusses customizing the special function and browse key names.

3.2 Configuring MS-Windows for NPL

While MS-Windows is much easier for the user to work with than MS-DOS, the configuration of MS-Windows and MS-Windows applications is more complicated. This section is intended to aid in the configuration of MS-Windows for NPL and NPL based applications.

3.2.1 WIN.INI Parameters

The WIN.INI file is a file used by MS-Windows to set various configuration parameters for the operation of MS-Windows on the host system. This file is edited dynamically by MS-Windows as changes are made to the MS-Windows configuration (i.e., the default file is changed, new program icons added, etc.). As such, no direct modifications are necessary for the Niakwa programs. For more information on this file, refer to the MS-Windows documentation.

3.2.2 DOS SHARE

The MS-DOS SHARE program must be loaded on any PC where multiple tasks are accessing local disk files, print devices, or diskette devices. The MS-DOS SHARE command should be loaded from the AUTOEXEC.BAT file. This may be done by adding the following statement to the AUTOEXEC.BAT file:

```
X:\DOS\SHARE
```

where X: is the drive and \DOS\ is the directory where the SHARE.EXE program is located.

NOTE: A warning message appears if an attempt is made to access local resources from multiple RunTime windows if the MS-DOS SHARE program is not loaded.

SHARE must also be used with Novell NetWare workstations when multiple tasks may be concurrently accessing:

- Local disk files.
- On-line (local) LPTx devices.
- Local serial devices.

SHARE is not required with Novell NetWare for concurrent access to:

- Disk files on the file server.
- The Novell NetWare spooler from the MS-Windows Print Manager.
- The same CAPTURED LPTx device.

Refer to Section 5.3 and 5.4 for more details on using the SHARE command with printing under MS-Windows.

NOTE: Microsoft acknowledges that using SHARE under MS-Windows, although required in many situations, can cause unpredictable results. This is a problem related to MS-Windows and not the NPL RunTime. Developers should be aware that when using SHARE, unexpected results could occur.

3.2.3 SMARTDrive

The Microsoft SMARTDrive (SMARTDRV.EXE) disk cache is installed and configured automatically during MS-Windows 3.1 installation. SMARTDrive and other disk caching software can significantly improve the performance of many MS-Windows functions, but is application dependent. SMARTDrive can easily be removed by adding a "rem" at the beginning of the line(s) that load it from the CONFIG.SYS and/or AUTOEXEC.BAT file.

HINT: Niakwa recommends that write caching always be disabled to preserve data integrity.

3.2.4 Installing Niakwa's Fonts

If use of either of the Niakwa-supplied font files, BASFONTS.FON or IBMFONTS.FON, is desired, the file must be installed as a MS-Windows font. Refer to Section 5.6.1 of this addendum for details.

3.2.5 Virtual Memory and Swap Files

When running the MS-Windows RunTime in 386 Enhanced Mode, additional memory for use by RunTime partitions (or any MS-Windows program) can be obtained by the use of swap files. Swapping involves moving information between memory and the hard disk to make room in memory for other information. This allows MS-Windows to use more memory than the amount of RAM that is physically installed in the host system.

Use of swap files is transparent to the NPL application. However, significant performance degradation can occur if swapping is used.

NOTE: The use of swap files is also referred to as virtual memory. Refer to the MS-Windows documentation for details on the use of swap flies.

3.2.6 Memory Management

If it is necessary to use the standard NPL MS-DOS RunTime with the /U option for use of UMBs on an 80386 system that is setup for use with the MS-Windows RunTime, the Microsoft HIMEM.SYS driver supplied with MS-Windows must be replaced with a supported 386 memory manager such as QEMM 386. Refer to Sections 4.4.12 and 8.4 of the MS-DOS Supplement for details.

3.2.7 MS-Windows Mode of Operation

Niakwa strongly recommends that the NPL MS-Windows RunTime be used in 386 Enhanced Mode. Standard Mode is supported, but 386 Enhanced Mode is preferred for the best performance of the NPL RunTime and its features.

3.3 Adding NPL Tasks to the Program Manager

Niakwa recommends adding a RunTime Program Icon to the Program Manager for each RunTime task that is to be run under MS-Windows. By adding the RunTime Program Icon to a group window, the operator can execute the RunTime quickly and easily. Refer to Section 4.3 for details on starting the MS-Windows RunTime.

The developer can add as many copies of the RunTime program icon to a Group Window as necessary; however, Microsoft recommends that a group should contain no more than 40 program icons. For example, a developer may want a separate program icon for each program or application on the system.

3.3.1 Command Line

The example below describes adding RunTime tasks to a program group window of the MS-Windows Program Manager.

To add a program icon to a program group, follow these steps:

1. Select the group to which the program icon is being added. The program group must be created if it does not exist (refer to the MS-Windows documentation on creating program groups).

2. From the menu bar, choose the File Menu.

3. Select Program Item from the New Program Object dialog box.

4. Select OK or press Enter. The Program Item Properties dialog box displays as shown in Figure 3-1.

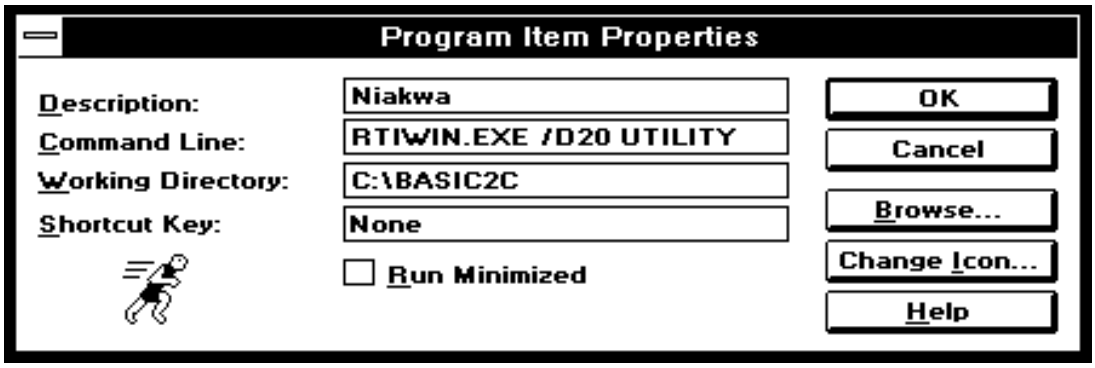


Figure 3-1

5. Select the Description text box and enter the title to appear under the icon.

6. Select the Command Line text box. If the path of the MS-Windows RunTime directory is specified in the Working Directory text box, (refer to Step 7, below) it is not necessary to specify the full drive and path name of the RunTime executables. In this case, only the executable name (RTIWIN or RTPWIN) is necessary. If no entry is made in the Working Directory text box, the full drive and path name must be specified in the Command Line text box. After the path name and/or RunTime executable name, add the command line parameters and application program boot name. For example, to create an icon to automatically load the Niakwa Utilities and allow for 20 device table entries, where the working directory is C:\BASIC2C the following should be entered in the Command Line text box:

```
RTIWIN /D=20 UTILITY
```

NOTE: Specifying the full path name for the RunTime program (RTIWIN.EXE or RTPWIN.EXE), whether in the Command Line or Working Directory text box, does an implicit change directory to the directory specified in the path. Because of this, it is not necessary to specify the full path for the BOOT program name if it resides in the same directory as the RunTime program.

7. An entry in the Working Directory text box is not required, but is recommended to simplify the Command Line entry. The path name specified in the Working Directory box is used to set the current directory prior to execution of the RunTime.

8. No entry is required for the Shortcut Key text box.

9. MS-Windows automatically selects the default RunTime icon unless the operator uses the Change Icon option. Refer to Section 3.3.2 below or the MS-Windows documentation for more information on choosing different icons (Niakwa provides three icon options).

9. Select OK to create the program icon. The new icon appears in the Main Group.

If the program icon must be redefined, access the File menu from the MS-Windows Menu Bar and select the Properties option. A program properties box appears. Make any necessary changes and Choose OK or press Enter.

Associated Boot Programs

It may also be useful to set up an "association" between the NPL Boot programs (with a .OBJ extension) and the MS-Windows RunTime. This may be done with the MS-Windows File Manager using the File Associate command. Refer to the MS-Windows documentation of the File Manager for details.

For example, to associate a boot file with the MS-Windows Non-interpretive RunTime, follow these steps:

1. Activate the MS-Windows File Manager.

2. Select the directory window containing the boot programs to be associated with the RunTime.

3. Highlight any .OBJ file as the file to be associated with the RunTime.

4. Select the File Associate option.

5. Enter the full path and name of the RunTime program to be associated with the boot program in the text box (i.e., C:\BASIC2C\RTPWIN.EXE).

6. Select OK or press Enter.

Once this association is set up, command lines given to the MS-Windows Program Manager may specify the full name of the .OBJ file (including the .OBJ extension, followed by any required options).

3.3.2 Choosing Icons

There are two icons that may be selected when defining an MS-Windows RunTime task. The first is the icon that appears on the Program Manager screen, the second is the icon that appears when the task is open and minimized. These two icons need not be the same.

The icon that appears with the Program Manager is selected by the Properties options when the task is originally setup (refer to Section 3.3.1). To change this icon, return to the Properties dialog box and use the Change Icon option as documented in the MS-Windows documentation.

NOTE: Niakwa provides three icons. These include: a Running man, Gold Key, and a Niakwa logo swiggle.

The icon the RunTime uses when a RunTime task is minimized must be changed by using the IconResourceFile and **IconNumber** options in the RTIWIN.INI file. This icon caption and the task window's title area are controlled by the **Caption** option in the RTIWIN.INI file.

NOTE: Additional icon figures are available from many third party MS-Windows developers or may be designed by using the MS-Windows Software Development Kit if the three provided by Niakwa are not adequate.

3.4 RTIWIN.INI

The RTIWIN.INI file provides the configuration link between NPL and MS-Windows. The customization of this file is extremely important in the setup of all NPL applications to be run in the MS-Windows environment. The following section provides a detailed description of the setup, organization, editing, and options available.

3.4.1 Setup

The RTIWIN.INI file is created automatically (in the directory where MS-Windows is installed), if it does not exist, when the MS-Windows Niakwa RunTime is started the first time. However, there are several options contained in this file that the developer can customize for particular application setup and use.

For network installations, it is important that each user has an individual copy of the RTIWIN.INI file in the user's own Windows directory (MS-Windows requires that each user have an individual working directory on network installations). Parameters that are common to all users may be stored in a network-wide RTIWIN.INI file. The location of this file is specified with the **NetworkIniFile** parameter in each user's RTIWIN.INI files. This file is never modified by the MS-Windows RunTime.

NOTE: Options specified in the local RTIWIN.INI file take precedence over options specified in the network-wide RTIWIN.INI file.

3.4.2 Organization

The RTIWIN.INI file is divided into two distinct descriptive areas. These include:

General Section:	Used to specify options that are general to all tasks for the user.
Individual Section:	Used to specify the option for each Niakwa application boot program.

These sections are automatically created if they do not exist. Some options in the individual sections are modified by the RunTime based on actions taken by the user. For example, resizing or relocating the window for the task causes the window's parameters to be updated. This allows the RunTime to "remember" changes made by the user and use the last selections as defaults the next time the task is executed.

The following example illustrates a typical RTIWIN.INI file.

```
[GENERAL]
NiaksecVersion=D
WarnUnreferencedIni=1
NetworkIniFile=K:\WINDOWS\RTIWIN.INI
[C:\BASIC2C\UTILITY.OBJ]
Window=22 87 594 479 0
SFKeys=280 150 0
Browse=184 -1 0
Follow=1
AutoSize=1 7 13 80

[C:\BASIC2C\DIAGBOX.OBJ]
Caption=Diagnostics
IconResourceFile=PROGMAN.EXE
IconNumber=5
Window=407 301 630 441 0
SFKeys=40 379 0
Browse=256 327 0
Follow=0
AutoSize=0 7 13 80
SFKeysCapNamesFile=K:\WINDOWS\MYWINDIR\MYSFKEYS.DAT
BrowseCapNamesFile=K:\WINDOWS\MYWINDIR\MYBROWSE.DAT
```

3.4.3 Editing

The RTIWIN.INI file may be customized by using any of the parameters discussed in Section 3.4.4. Editing can be performed with any text editor (i.e., MS-Windows Notepad, EDIT, EDLIN, etc.).

NOTE: It is strongly recommended that, if this file is to be edited manually, the WarnUnreferencedIni option be used to check for entry mistakes.

3.4.4 Parameters

The following is a complete description of the various RTIWIN.INI parameters. If a parameter is not specified, the default value, as described for each parameter below, is used.

NOTE: The options specified in this file apply both to the Interpretive (RTIWIN.EXE) and Non-interpretive (RTPWIN.EXE), RunTime programs--there is not a separate RTIWIN.INI file for each.

AutoSize

Purpose: Allows specification of the status of the autosize flag (on or off) and the font sizes to be used if off. In addition, the maximum width of the screen used by the application is specified (usually 80 columns, but may also be 132 for applications that select a wide display).

The **Autosize** option and font size may be changed by the user by resizing the window. The values are saved in the application's option section when the window is closed as the defaults for the next session.

Values are, in order:

1. 0 for off or non zero for on
2. Font width in pixels when Autosizing is off
3. Font height in pixels when Autosizing is off
4. Application display size in columns (usually 80 or 132)

Value: Four numeric values separated by blanks

Default: 1 0 0 80

Example: AutoSize= 0 6 9 80

BrightBackground

Purpose: A display preference option that specifies whether background colors in the RunTime window should be bright.

Value: Numeric, 0= normal background, not 0= bright

Default: 0

Example: BrightBackground= 1

Browse

Purpose: A display option that specifies where and if the Browse Key window should appear. Three numbers are required, specifying the x and y co-ordinates (in pixels) from the top left of the screen, and a third number showing whether the window is initially visible.

NOTE: If the number of pixels is out of range for the display being used (i.e., set to 790 when the monitor can only display 748 pixels horizontally), the window is automatically moved to make it at least partially visible.

The location and visibility of the window may be changed by the user. The values are saved in the application's RTIWIN.INI option section when the window is closed as the defaults for the next session.

Value: 3 Numeric,
1. Specifies x pixels from the top left of the screen
2. Specifies y pixels from the top left of the screen
3. 0= don't display initially, not 0= display initially

Default: 0 0 0

Example: Browse = 448 284 1

BrowseCapNamesFile

Purpose: A pop-up preference option that specifies a file name that contains replacement labels for the Browse Key window display. The format of this file is detailed in Section 3.5 of this chapter.

Value: String, full path name of file.

Default: Null string= no replacement

Example: BrowseCapNamesFile= C:\WINDOWS\MYBROWSE.TXT

Caption

Purpose: A display option that specifies the string that appears in the RunTime window's caption (title) area. The default value is derived from the name of the boot program used to start the application.

Value: String

Default: Derived from boot program

Example: Caption= Niakwa Utilities

NOTE: If no boot program is specified, the default boot program (BOOT.OBJ) appears as the RunTime window caption.

CommFlushDelay

Purpose: Specifies the delay time (in seconds) for characters to leave the serial port output buffers before generating a timeout error.

Value: Numeric

Default: 30

Example: CommFlushDelay= 30

CommInputBufferSize

Purpose: Specifies the size in bytes required for the input buffers used when accessing serial ports.

Value: Numeric

Default: 512

Example: CommInputBufferSize= 1024

CommOutputBufferSize

Purpose: Specifies the size in bytes required for output buffers used when accessing serial ports.

Value: Numeric

Default: 512

Example: CommInputBufferSize= 1024

DllDupDirectory

Purpose: When using external libraries, which are non-shareable, the RunTime must make temporary copies of the DLL. This option specifies where the temporary copies should be located.

NOTE: This option may only appear in the [General] section of the RTIWIN.INI file.

Value: Drive and directory

Default: . (same drive and directory as the original DLL).

Example: DllDupDirectory= C:\DLLTEMP

ExclusiveWhenNetworkLocksHeld

Purpose: Switching to another window while network file locks are held by NPL causes other network users waiting for the same files to wait longer for these resources. If the window which is switched to run a task in exclusive mode issues a system modal dialog box, this delay can be considerable.

For applications that lock network files for brief durations, setting this flag to 1 causes the RunTime to run in exclusive mode while network locks are held.

NOTE: The application may override this option by setting byte 43 of $OPTIONS.

Value: Numeric

Default: 0

Example: ExclusiveWhenNetworkLocksHeld= 1

ExternalLibrary

Purpose: Allows the specification of the names of the external libraries (DDL's) that should be loaded upon execution of the application.

The option should be in the general or application section of the RTIWIN.INI file.

The value of x is an integer index number. If more than one ExternalLibraryx = option is entered in a section, the index (x value) must start at 1 and be consecutive. Refer to Section 8.5 for more information on loading multiple DLL's.

Value: String

Default: Null

Example: ExternalLibrary1= C:\BASIC2C\NPLWIN.DLL
ExternalLibrary2= C:\BASIC2C\NPLDVS.DLL

Follow

Purpose: Specify whether scroll bars should adjust automatically to keep the cursor visible when awaiting input. This option may be changed by the user from the MS-Windows menu bar.

The status is saved in the application's RTIWIN.INI option section when the window is closed as the default for the next session.

Value: Numeric, 0= do not follow cursor, not 0= follow cursor

Default: 0

Example: Follow= 1

FontCharSet

Purpose: A display option that specifies the character set type used in the Run-Time window. This option is typically used only when using a font file other than BASFONTS.FON. The value specified must correspond to the character set type specified in the font file to be used. The actual character set used may vary depending on available fonts.

Changing the font character set effects the available character set displayed by the NPL. In particular, the NPL graphic characters may not be available unless the provided Niakwa fonts are used.

Defined Value:
0 ANSI
2 Symbol
178 NPL character set (BASFONTS.FON)
179 IBM character set (IBMFONTS.FON)
255 OEM

Value: Numeric

Default: 178 (BASFONTS.FON)

Example: FontCharSet= 179

FontFaceName

Purpose: A display option that specifies the name of the font used in the Run-Time window.

This should be the name of a fixed pitch font. The actual font used may vary depending on the available fonts. To select a particular font, it may also be necessary to change the **FontCharSet** option.

Value: String

Default: BASIC2C

Example: FontFaceName= IBASIC

HaltRequestPeriod

Purpose: **HaltRequestPeriod** is a performance tuning parameter that specifies the number of times the RunTime can "branch" within a program before checking for MS-Windows messages. Decreasing the value increases the responsiveness of the Niakwa RunTime to MS-Windows messages. Increasing the value improves performance by reducing the amount of time the Niakwa RunTime spends checking for MS-Windows messages, but may reduce responsiveness to operate actions.

NOTE: The default value of 100 is suitable for most applications.

"Branches" are defined as any GOTO, NEXT, or RETURN statement or falling through from one program line to the next.

Windows "messages" are used by the RunTime for:

- The HALT Key(CTRL/BREAK)
- Mouse-Input
- Window resizing

HaltRequestPeriod is also used by the Niakwa RunTime to determine how often to synchronize the screen display. Because scrolling speed under MS-Windows is fairly slow, the MS-Windows version of the RunTime performs optimization on screen output. Any output that would be scrolled without waiting for operator input (KEYIN, LINPUT, INPUT) is buffered until the next **HaltRequestPeriod** expires. At that time, the screen is updated with current contents of the internal screen buffer. The effects of this buffering can be seen by entering a LISTF command with a lengthy program in memory.

Any keyboard input, disk I/O, or $BREAK statements cause the RunTime to terminate the current **HaltRequestPeriod** and check for MS-Windows messages and synchronize the screen.

Value: Numeric, maximum 65535, 1= check for HALT at each branch.

Default: 100

Example: HaltRequestPeriod= 50

IconNumber

Purpose: If **IconResourceFile** is specified, this option defines which icon from the resource file is displayed when the window is minimized. If the resource file contains multiple icons (i.e., a .EXE or .DLL file), the option **IconNumber** must be set to the number of the icon that is wanted (starting at 1). If the resource file contains only one icon (i.e., a .ICO file) the **IconNumber** must be set to 0.

Value: Numeric

Default: 0

Example: IconNumber= 1

IconResourceFile

Purpose: This option allows for selecting an icon from a resource file that is to be displayed when the window is minimized. If the resource file contains multiple icons (i.e., a .EXE or .DLL file), the option **IconNumber** must be set to the number of the icon that is wanted (starting at 1). If the resource file contains only one icon (i.e., a .ICO file) **IconNumber** must be set to 0.

Value: String, full path name of file

Default: Null (use standard icon)

Example: IconResourceFile= PROGMAN.EXE

LockRetryDelay

Purpose: Specifies the time to wait (in milliseconds) before retrying if a file lock request is refused by the file server. Refer to Section 5.2.1 for more details.

Value: Numeric

Default: 1000

Example: LockRetryDelay= 0

LockWaitTimeout

Purpose: Specifies the time to delay at the file server (in timer ticks, 18= 1 second) waiting for a locked file to be released before reporting the file is locked. Refer to Section 5.2.1 for more details.

Value: Numeric

Default: 18

Example: LockWaitTimeout= 90

MouseClickKeys

Purpose: Allows redefinition of the key values returned to an application when the mouse keys are clicked, released, or double-clicked. Six values may be specified, separated by blanks or commas.

Value: Six key values, separated by blanks or commas.

In order, the key values specify replacement values for:

Default	**Key value**
’F1	(1) Left mouse button pressed
’F2	(2) Left mouse button released
’F3	(3) Left mouse button double-click pressed
’F4	(4) Right mouse button pressed
’F5	(5) Right mouse button released
’F6	(6) Right mouse button double-click pressed

An empty value (shown by commas with no value between them) leaves the generated value as the default. Values must be hexadecimal (special function values should be preceded by a "’"). The non-special key value, 0 is reserved to show that no key is generated for this mouse event.

Default: Null (use defaults for all mouse click keys).

Example: MouseClickKeys= 0,0,82,0,0,’7E

MouseDragNSEWKeys

Purpose: Allows redefinition of the key values returned to an application when the mouse is dragged (moved while a key is pressed). Four values may be specified, separated by blanks or commas.

Value: Four-key values, separated by blanks or commas.

In order, the key values specify replacement values for:

Default	Key value
'F7	(1) Mouse dragged North (up)
'F8	(2) Mouse dragged South (down)
'F9	(3) Mouse dragged East (right)
'FA	(4) Mouse dragged West (left)

An empty value (shown by commas with no value between them) leaves the generated value as the default. Values must be hexadecimal (special function values should be preceded by a "'"). The non-special key value, 0 is reserved to show that no key is generated for this mouse event.

Default: Null (use defaults for all mouse drag keys).

Example: MouseDragNSEWKeys= '46,'45,'4C,'4D

NetworkIniFile

Purpose: Initialization values for applications may be set up in a common file that is referenced by all users on a networked system. The name of the network configuration file is specified by this option. This file is not changed by the MS-Windows RunTime and options in the local RTIWIN.INI file take precedence over the network file.

NOTE: This option should only appear in the [General] section of the local RTIWIN.INI files.

Value: String, full path name of file.

Default: Null (no network-wide options).

Example: NetworkIniFile= K:\WINDOWS\NWRTIWIN.INI

NetworkPartitionsFile

Purpose: When used with TERMINAL.TBL (which generates unique #TERM values for each node on a network), this option allows generation of unique values for #PART for each window on a network wide basis. The option specifies the name of a configuration file for the application. This configuration file limits access to the application to a limited number of values of #TERM and #PART (as they would be set without this option). Each line of the specified file must have the format:

```
#TERM-value #PART-value [;comment]
```

The RunTime assigns a new #PART value by searching for an entry containing the original #TERM value (as generated by TERMINAL.TBL) and the original #PART value (as generated by the order in which tasks are executed on a given workstation and as modified by use of the **ReservedPartitions** and **Partition** options of RTIWIN.INI). The new #PART value assigned is the entry number of the matching entry in the **NetworkPartitionsFile**. A task matching entry one is assigned a #PART value of 1, a task matching entry 2 is assigned a #PART value of 2, and so on.

The name for the this file is generated by use of the **NetworkPartitionsFile** option. If no value is set for **NetworkPartitionsFile**, remapping of the #PART values does not occur.

If **NetworkPartitionsFile** is used, only tasks whose original #TERM and #PART values appear in the specified file are permitted to run. If the specified file is not found, or a matching entry is not found or the Partition number has already been mapped to another Partition number, the RunTime is terminated with a message "Cannot Determine Partition Number".

Use of **NetworkPartitionsFile** is only meaningful on Novell Netware networks where a TERMINAL.TBL file is in use (otherwise, #TERM is always 1).

Different **NetworkPartitionsFile** files may be used for different applications on the same system.

HINT: To ensure consistent assignment of the **NetworkPartitionsFile** name, it is recommended that this option be specified in the **NetworkIniFile** rather than the local RTIWIN.INI file.

Any #TERM remapping performed by the **TerminalIsPartition** option occurs after #PART remapping is performed by **NetworkPartitionsFile**.

The standard MS-DOS/Novell RunTime does not use the **NetworkPartitionsFile**. To ensure unique #PART generation for installations where some workstations are using the MS-Windows RunTime version and some workstations are using the standard MS-DOS/Novell NetWare RunTime version, two steps are necessary:

1. Specify **ReservedPartitions= 1** either in the **NetworkIniFile** or in each of the local RTIWIN.INI files. This ensures that the original #PART generated for each MS-Windows task is always 2 or greater.

2. At the start of the **NetworkPartitionsFile**, place one entry for each workstation, specifying the #TERM value for the workstation and a #PART value of 1. This, in effect, reserves the first X #PART values for MS-DOS tasks. The #PART values reserved are the same as the #PART values generated by TERMINAL.TBL.

On large networks, it is easy to generate high #PART and #TERM values. If #PART or #TERM values are greater than 99, $PSTAT will not correctly reflect the #TERM value.

NOTE: #PART and #TERM values greater than 999 cannot be generated.

Refer to Section 6.2 for further information and guidelines for establishing unique task identification. Refer to Section 6.2.3 for a detailed example in setting up the NetworkPartitionsFile.

Value: String, full path name of file.

Default: Null (no network partitions file).

Example: NetworkPartitionsFile= K:\BASIC2C\NETPARTS.TBL

NiaksecVersion

Purpose: In situations where the standard security TSR program (NIAK-SECA.COM) cannot be used because interrupt 61H is already in use, this option must be used to indicate that a different version (NIAK-SECB.COM - NIAKSECF.COM) is used instead.

NOTE: This option may only appear in the [General] section of the RTIWIN.INI file.

Value: Letter value (A, B, C, D, E, or F) of version used.

Default: A (use standard program)

Example: NiaksecVersion= B

ParallelFullDelay

Purpose: Specifies the number of time units (in milliseconds) to delay when LPT drivers indicate that transmit buffers are full before sending more data.

Value: Numeric

Default: 200

Example: ParallelFullDelay= 10

ParallelRetryCount

Purpose: Specifies the number of times to ask LPT drivers to send data before accepting an indication from the driver that transmit buffers are full.

Value: Numeric

Default: 20

Example: ParallelRetryCount= 10

Partition

Purpose: In situations where a specific application must always have the same partition number, it may be directly assigned using this option. If the option is not specified, the first available partition number is allocated.

To ensure that partition numbers allocated by this option are not used by other applications, use the ReservedPartitions option.

NOTE: This option normally does not appear in the [General] section of the RTIWIN.INI file.

Partition numbers generated by the **Partition** parameter may be remapped by use of the **NetworkPartitionsFile** parameter. Refer to Section 6.2 for more information and guidelines on unique task identification.

Value: Numeric starting at 1 to 999.

Default: None (first free non-reserved partition dynamically assigned)

Example: Partition= 2

PerimiterIsBackground

Purpose: A display preference option that specifies whether the perimeter area of the RunTime window (area outside normal display area) should always be the same as the default background color (overriding any value specified by program).

Value: Numeric, 0= program specifies perimeter, not 0= use background color

Default: 0

Example: PerimiterIsBackground= 1

PrinterConfig

Purpose: If a $DEVICE specification in the form of "> (@printerid)" is used, the value of this option is used as an indirect print spooler specification. The printer id may be any legal string, but recommended values are numbers starting at 1.

The format of this option is the same as that for the WIN.INI "device= " option (i.e., three strings separated by commas) specifying **DeviceName, DriverName, Output** in that order. **DeviceName** specifies the specific device to be supported. **DriverName** specifies the MS-DOS filename (without extension) of the device driver. **Output** specifies the MS-DOS file or device name for the physical output medium. Refer to the MS-Windows documentation on the MS-Windows Print Spooler and Section 5.4 of this Addendum for more information.

Value: Three strings separated by commas.

Default: None

Example: PrinterConfig1= HP ThinkJet (2225 C-D),THINKJET,LPT1:

ReservedPartitions

Purpose: In situations where specific applications must always have the same partition number, it is usually desirable to reserve these partition numbers so applications that use dynamically assigned partition numbers do not conflict.

This option specifies the partition numbers (1 to n) that are never assigned dynamically (i.e., only are used by applications with a **Partition** option in the RTIWIN.INI file).

NOTE: This option may only appear in the [General] section of the RTIWIN.INI file and must be used with the Partitions option.

Value: Numeric 0-999

Default: 0

Example: ReservedPartitions= 2

SFKeys

Purpose: A display option that specifies where and if the SF Keys window should appear. Three numbers are required, specifying the x and y co-ordinates (in pixels) from the top left of the screen, and a third number showing whether the window is initially visible.

NOTE: If the number of pixels is out of range for the display being used (i.e., set to 790 when the monitor can only display 748 pixels horizontally), the window is automatically moved to make it at least partially visible.

The location and visibility of the window may be changed by the user. The values are saved in the application's option section of the RTIWIN.INI file when the window is closed as the defaults for the next session.

Value: 3 Numeric

1. Specifies x pixels from the top left of the screen
2. Specifies y pixels from the top left of the screen
3. 0= don't display initially, not 0= display initially

Default: 0 0 0

Example: SFKeys= 0 403 0

SFKeysCapNamesFile

Purpose: A display preference option that specifies a file name that contains replacement labels for the SF Keys window display. The format of this file is detailed in the Section 3.5 .

Value: String, full path name of file.

Default: Null string= no replacement

Example: SFKeysCapNamesFile= C:\WINDOWS\MYSFKEYS.TXT

SFKeysStyle

Purpose: A display preference option that specifies which style of SF Keys window should appear.

Value: Numeric, 0= one row of 16 keys, 1= two rows of 8 keys

Default: 0

Example: SFKeysStyle= 1

ShareWarning

Purpose: To suppress the warning message box when the MS-Windows Run-Time detects that the MS-DOS SHARE command is not running and a second MS-Windows RunTime attempts to access local files (i.e., non-network files).

WARNING--This option should never be used if file or device sharing is to be used.

Value: Numeric,

0= suppress SHARE warning,
1= display SHARE warning box

Default: 1

Example: ShareWarning= 0

StandardColorRGB0 to StandardColorRGB15

Purpose: Allows explicit definition of the 24-bit RGB (red-green-blue) values used by the MS-Windows RunTime to request colors from the MS-Windows display driver. The default values are typically suitable for drivers (VGA, EGA) which can display at most 16 solid colors.

On displays with more color capability, this option allow selection of different shades of color.

If non-solid colors are specified, MS-Windows automatically uses the closest solid color.

The standard colors are normally:

0 black	8 gray (= bright black = dark gray)
1 blue	9 bright blue
2 green	10 bright green
3 cyan	11 bright cyan
4 red	12 bright red
5 magenta	13 bright magenta
6 brown	14 bright brown (yellow)
7 white (= light gray)	15 bright white

HINT: If the colors are remapped, remember the bright video attributes are obtained by adding 8 to the color value of the foreground. Monochrome displays use standard color 0 for black and 15 for white.

Each entry requires 3 numbers, which should each be in the range 0-255, separated by spaces. The values specify the intensity of each of the color values red, green and blue (in that order). The custom color display in the MS-Windows Control Panel may be used to determine appropriate RGB values.

Value: 3 Numerics in range 0-255

Default:

StandardColorRGB0	0	0	0
StandardColorRGB1	0	0	128
StandardColorRGB2	0	128	0
StandardColorRGB3	0	128	128
StandardColorRGB4	128	0	0
StandardColorRGB5	128	0	128
StandardColorRGB6	128	128	0
StandardColorRGB7	192	192	192
StandardColorRGB8	85	85	85
StandardColorRGB9	0	0	255
StandardColorRGB10	0	255	0
StandardColorRGB11	0	255	255
StandardColorRGB12	255	0	0
StandardColorRGB13	255	0	255
StandardColorRGB14	255	255	0
StandardColorRGB15	255	255	255

Example: StandardColorRGB1= 0 0 192

NOTE: The default colors used by MS-Windows can, on some configurations, cause various attribute combinations to become invisible. In particular, REVERSE/BRIGHT is often affected by this. Remapping the colors may resolve this.

TerminalIsPartition

Purpose: To allow #TERM values to be set equal to #PART values.

NOTE: This change is made after any remapping of partitions caused by the NetworkPartitionsFile option.

Refer to Section 6.2 for more information and guidelines on unique task identification.

Value: Numeric, 0= do not change #TERM, 1= set #TERM from #PART

Default: 0

Example: TerminalIsPartition= 1

WarnUnreferencedIni

Purpose: This option checks for possibly misspelled or duplicate values in the RTIWIN.INI file. If set to a non-zero value, key values of the RTIWIN.INI file in the general or application section that are not known to the RunTime generate warning messages at start-up time.

Value: Numeric 0= no warnings, not 0= display warnings

Default: 0

Example: WarnUnreferencedIni= 1

Window

Purpose: A display preference option that specifies where and how large the RunTime window should appear. Five numbers are required, specifying the x and y coordinates (in pixels) from the top left of the screen, the x and y size of the screen (in pixels) and a fifth number showing whether the window is initially expanded to full screen size.

NOTE: If the number of pixels is out of range for the display being used (i.e., set to 790 when the monitor can only display 748 pixels horizontally), the window is automatically moved to make it at least partially visible.

The location, size and visibility of the window may be changed by the user. The values are saved in the application's RTIWIN.INI option section when the window is closed as the defaults for the next session.

Value: 5 Numeric

1. Specifies x pixels from the top left of the screen
2. Specifies y pixels from the top left of the screen
3. Specifies x horizontal size in pixels from 1.
4. Specifies y vertical size in pixels from 2.
5. 0= don't display full screen, non 0= display full screen

Default: Defaults provided by system

Example: Window= 1 3 640 271 0

3.5 Customizing Special Function/Browse Key Names

These files allow the developer to customize the key labels as they appear on the Browse and SF Key windows described in Section 4.3. The following is a discussion of the option for these files.

These files are ASCII text files, created by any text editor (i.e., the MS-Windows Notepad, EDIT, EDLIN, etc.), to specify the replacement information for the default labels.

Each line may have one of the following formats:

1. To specify the source file is OEM character set, include the line oem as shown below.

   ```
   oem
   ```

   This line must precede any lines that require character set translation to ANSI.

2. To specify source file is ansi character set, include the line ansi as shown below.

   ```
   ansi
   ```

   This line may precede any lines that do not need character set translation.

3. To add comments to a line, add a ";" before the comments as shown below.

   ```
   ;optional comment
   ```

4. To specify a new caption for the window name, add a line that sets the caption equal to a new string as shown below.

   ```
   Caption="string"              ;optional comment
   ```

5. To specify a new label for a key, add a line renaming the label as shown below.

   ```
   OldLabel="string"             ;optional comment
   ```

6. To specify a new label and a new non-function key value, add a line renaming the label and defining the key in hex as shown below.

   ```
   OldLabel="DEL"=7F             ;optional comment
   ```

7. To specify a new label and new function key value, add a line renaming the label and defining the key in hex as shown below.

```
OldLabel="QUIT"='7F          ;optional comment
```

8. To specify a new label and new function key values for unshifted and shifted values, add a line renaming the label and defining the key in hex as shown below.

```
OldLabel="QUIT"='7E,'7F      ;optional comment
```

NOTE: The "OldLabel" value must match the standard label for the key in the uncustomized dialog box (e.g., in SF Keys labels are '0, '1, etc.).

The following examples illustrate actual replacement files.

Example 1 - SF Keys:

```
'0= "Add"
'1= "Sub"
'2= "Mult"
'3= "Div"
'4= "Total"
'5= "Calc"
'6= "Del"
'7= "Help"
```

Example 2 - Browse Keys:

```
Cancel= "Quit"
Prev= "Pg Up"
Next= "Pg Dn"
Insert= "Add"
Delete= "Subtract"
Exec= "Run"
Back= "Reverse"
Space= "Blank"
Return= "Enter"
```

CHAPTER 4

RUNTIME OPERATION

4.1 Overview

This chapter discusses the operation of the Niakwa MS-Windows RunTime.

Section 4.2 discusses the various MS-Windows modes and their effect on the MS-Windows RunTime.

Section 4.3 discusses starting the MS-Windows RunTime.

Section 4.4 discusses the menu bar options available to all MS-Windows RunTime tasks (Browse Keys, SF Keys, Follow, and Autosize) and HELP Processor options

Section 4.5 discusses minimizing a RunTime task.

Section 4.6 discusses resizing or moving a RunTime window.

Section 4.7 discusses the RunTime startup options that are specific to MS-Windows.

Section 4.8 discusses the various methods of closing a RunTime task.

Section 4.9 discusses using the standard Niakwa RunTime.

4.2 MS-Windows Modes

The MS-Windows RunTime can be executed in either MS-Windows Standard or 386 Enhanced Mode. Refer to the MS-Windows documentation for more information on the various MS-Windows modes and their requirements and start up options.

4.3 Starting the RunTime

The MS-Windows RunTime can be started from the MS-Windows Program Manager or from a command line using the MS-Windows Run Program feature.

NOTE: The NIAKSECx command must be executed prior to starting the MS-Windows environment. If NIAKSECx has not been run, exit MS-Windows and run the command. NIAKSECx is necessary for the RunTime to execute. Refer to Section 2.5 of this addendum for more information on the security for the MS-Windows RunTime.

4.3.1 Starting the RunTime from the Program Manager

To start the MS-Windows RunTime from the MS-Windows Program Manager, follow these steps:

1. Start the MS-Windows Program Manager if it is not currently displayed. The Main Group icons or the window that contains the RunTime icon should be displayed.

2. Select the RunTime icon to be run by double-clicking on the program icon to be run or highlighting it and pressing Enter. Refer to Section 3.3 for information on creating and installing the RunTime icon.

NOTE: If the Gold Key security has not been installed, the operator is prompted to insert the Gold Key Diskette. Enter the appropriate drive letter in the Text Entry Box and select the OK command button.

3. The MS-Windows RunTime loads into memory and starts executing the application boot program that was specified in the task command line.

NOTE: Refer to Appendix A of this addendum for a list of common problems if the Run-Time task does not execute properly.

4.3.2 Starting the RunTime from a Command Line

The MS-Windows RunTime can also be started from the File Run command in the Program Manager. To start the MS-Windows RunTime by using this option, follow the steps shown below:

1. Choose the File Run command from the Program Manager's File Menu. The Run dialog box appears as shown in Figure 4-1.

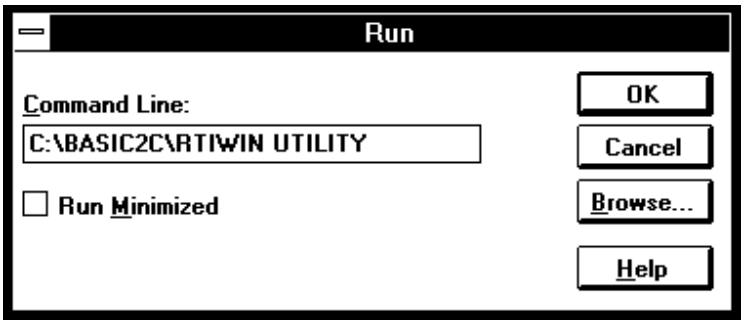


Figure 4-1

2. Enter the full path name of the MS-Windows RunTime. For example, if the RunTime is located in the C:\BASIC2C directory, the following would be entered in the text-entry box:

```
C:\BASIC2C\RTIWIN.EXE <options> <bootname>
```

where < options> represents any start-up options needed for the application and < bootname> is the application boot file.

3. Select OK or press Enter. The RunTime then loads and begins to execute the boot program specified in step 2.

4. If an entry exists in RTIWIN.INI for the specified boot file, the options specified in that section are used. Otherwise, a new section for the task is created in the RTIWIN.INI file automatically.

4.4 Menu Bar Options

The standard MS-Windows menu bar for the NPL RunTime for MS-Windows contains two pull down menus.

The Options Menu is discussed in Section 4.4.1 and the HELP menu is discussed in Section 4.4.2.

NOTE The menu bar can be modified by external DLLs. Refer to Chapter 8 for details.

4.4.1 Interactive Options

The RunTime provides four user-selectable options from the Option Menu accessible from the RunTime Menu Bar. These options are:

- Browse Keys
- SF Keys
- Follow
- Autosize

To activate any of these options, it is necessary to "check" it. To do this, first select the Option Menu from the menu bar (the Option Menu displays the four interactive options). To check an item, highlight the option and click the left button of the mouse once. When an option is selected, a checkmark appears next to the item to show that it is active. To deactivate an option, select it again and the checkmark is removed.

HINT: The Browse Keys and SF Keys options may also be deactivated by double-clicking the option bar on the appropriate option keys window.

NOTE: The default values are stored in the RTIWIN.INI file. Refer to Section 3.4 for more information on this file.

Any of these options can be used simultaneously. The following describe these options in detail.

NOTE: The RTIWIN.INI file is automatically updated with the status of these options when the RunTime task is closed. The next time the RunTime task is opened, it uses this information as its default.

Browse Keys

This option allows the NPL virtual keys to be accessed directly from the screen by using a mouse. When this option is active, a window appears on the screen with the names of the NPL keys that can be accessed with a mouse.

The default keys are set as: Cancel, Prev, Insert, Exec, Tab, Next, Delete, Back Space, Return, and the North, South, East and West directional keys as shown in Figure 4-2.

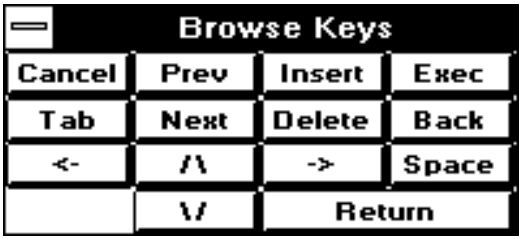


Figure 4-2

The key labels and caption displayed can be changed by the developer. This can be done by using the **BrowseCapNamesFile** option in the RTIWIN.INI file. Refer to Section 3.4 and 3.5 for more information.

Keys are selected by locating the mouse cursor to the desired key and clicking the left button. Holding the right button while pressing the left button produces the shifted value of the key (i.e., SHIFT-INSERT).

To move the Browse Key window to another position on the desktop, use the mouse to drag the window to the new location as if it were any other window. If the associated parent task for the Browse Key window is minimized, the Browse Key display is not visible as long as the task remains minimized.

NOTE: The status (active or not active) and the last location of the Browse window, before the RunTime task is closed, is automatically stored by the Browse option in the RTI-WIN.INI file. Refer to Section 3.4 for more information on this feature.

Special Function Keys

This option allows the NPL Special Function (SF) keys to be accessed directly from the screen by using a mouse. When this option is active, a window appears on the screen with the names of the NPL SF Keys that can be accessed with a mouse.

The default keys are set as SF Keys 0-15 in a single strip as shown in Figure 4-3.

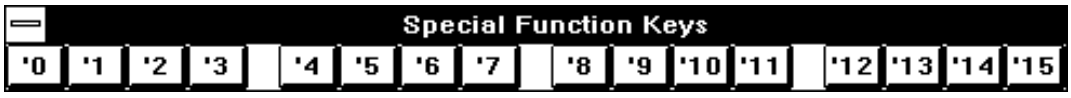


Figure 4-3

By using the **SFKeysStyle** option in the RTIWIN.INI file, the developer can split the function keys into two rows of 8 each as shown in Figure 4-4.

Figure 4-4

HINT: The key labels and caption displayed can be changed by the developer. This can be done by using the **SFKeysCapNamesFile** option in the RTIWIN.INI file. Refer to Section 3.4 and 3.5 for more information.

Keys are selected by locating the mouse cursor to the desired key and clicking the left mouse button. SF Keys 16-31 can be accessed through this option by holding the right mouse button and clicking the left mouse button when the mouse cursor is on the appropriate SF Key. For example, to generate the SF Key 17, press and hold the right mouse button and click the left button when over the SF Key 1.

To move the SF Keys window to another position on the desktop, use the mouse to drag the window to the new location as if it were any other window. If the associated parent task for the SF Keys window is minimized, the SF key display is not visible as long as the task remains minimized.

NOTE: The status (active or not active) and the last location of the SF Keys window, before the RunTime task is closed, is automatically stored by the SFKeys option in the RTIWIN.INI file. Refer to Section 3.4 for more information on this feature.

Follow

In cases where the visible window is too small to display the entire RunTime screen, the Follow option keeps the cursor visible in the RunTime task window. When this option is active, the window automatically scrolls to keep the cursor visible when awaiting input. When switched off, the cursor can move off the visible screen area displayed on the window to other parts of the full screen display (the MS-Windows scroll bars can be used to reposition the active display to show the cursor).

NOTE: The status of this option is saved by the Follow option in the RTIWIN.INI file when the current task is closed. The next time the RunTime task is started, it uses the saved status as its default. Refer to Section 3.4 for more information on this feature.

Autosize

When this option is selected, the MS-Windows RunTime selects the closest available font to fit a full screen (24 rows x 80 or 132 columns) into the current window size. Scroll bars only appear if the window size and font size combination do not allow the full Niakwa screen to appear in the resized window. When the option is not selected, changing the size of the window does not affect the font size. Scroll bars always appear when the full screen does not fit in the current size window with the font setting being used.

HINT: The best way to use this option is to select the option and resize the window until a comfortable font size is displayed. This font size can then be saved by switching the Autosize option to off. When switched off, the font size does not change even if the window is resized.

NOTE: The status of this option as saved by the AutoSize option in the RTIWIN.INI file when the current task is closed. The next time the RunTime task is started, it uses the saved status as its default. Refer to Section 3.4 for more information on this feature.

The following options of the RTIWIN.INI file also affect font size:

- Window
- FontFaceName
- FontCharSet.

4.4.2 Help Option

Selecting the Help Option on the menu bar displays a sub-menu with the following choices:

Topic	Invokes the NPL Help Processor. Selecting this menu item is equivalent to entering the HELP key from the keyboard. This item is available only if $HELP is set to a non-blank value. Refer to Chapter 11 of the NPL Programmer's Guide for details.
About RTI	Selecting this menu item displays a message box showing the current RunTime version in use.

4.5 Minimizing a Runtime Task

To minimize a RunTime task, select the minimize icon (refer to the MS-Windows documentation for detailed information).

NOTE: When a task is minimized it continues to execute, but any Browse or SF Key windows displayed are not visible as long as the task remains minimized.

The RunTime Program icon is used as a default icon by MS-Windows for the minimized task. The developer can choose another icon to be used when a RunTime task is minimized. This icon can be specified by using the **IconResourceFile** and **IconNumber** options in the RTIWIN.INI file. Refer to Section 3.4 for more information on this option.

4.6 Resizing or Moving Task Windows

RunTime tasks, Browse Keys, or SF Keys windows can all be moved by using the standard MS-Windows method of moving a window (select and drag). In addition, RunTime task windows can be resized using the standard MS-Windows method of resizing windows. The last size and position of these windows is saved by the appropriate option in the RTIWIN.INI file when the task is closed. The size and position parameters are then used as the default values the next time the RunTime task is started. Refer to Section 3.4 of this Addendum for more information.

4.7 RunTime Startup Options

All RunTime startup options available under the MS-DOS version are available under the MS-Windows versions except for those noted below. Refer to Section 4.4 of the NPL MS-DOS Supplement for a complete discussion of the startup options under the MS-DOS version of NPL.

4.7.1 /G Option

Under the MS-Windows version of NPL, graphics mode is always active. Therefore, the /G option has no effect on the MS-Windows RunTime operation.

4.7.2 /H Option

The /H option is not supported under the MS-Windows version of NPL. The dynamic nature of the MS-Windows environment allows the handle table to expand to the necessary size as required by the RunTime.

4.7.3 /K Option

The /K option is not necessary under MS-Windows. Mouse support is automatic under MS-Windows.

4.7.4 /M Option

The /M option is not supported under the MS-Windows version of NPL. Under MS-Windows, the RunTime has access to all available system memory.

4.7.5 /R Option

The /R option is not supported under the MS-Windows version of NPL.

4.7.6 /U Option

The /U option is not supported under the MS-Windows version of NPL. Under MS-Windows, the RunTime has access to available system memory.

4.7.7 /X Option

Under MS-Windows the /X option has been enhanced to allow more than one external library (DLL) to be specified. Refer to Chapter 8 for details.

4.8 Closing the RunTime Task

A RunTime task can be closed by using the Niakwa $END command. This can be issued in Interpretive mode or as part of the program code. The RunTime task can also be closed by using the MS-Windows "Exit Program" menu option.

The HEX(10) bit of byte 33 of $OPTIONS is used to instruct the MS-Windows RunTime to suppress the ability to kill RunTime tasks from MS-Windows. If this bit is set, then:

- The Close Task option from the MS-Windows menu bar is not available.
- Attempting to close the task using a double-click is not available.
- A warning is generated if the user attempts to kill the task from the Task Manager.
- A warning is generated if the user attempts to exit MS-Windows while a RunTime task is open.

NOTE: If the HEX(02) bit of byte 33 (suppress Kill RunTime, from the HELP Processor) is on, then suppression of the MS-Windows methods of killing the task is automatically on, whether or not the HEX(10) bit is set.

Refer to the NPL Statements Guide for more information on $OPTIONS.

HINT: This may be desirable to prevent users from accidentally exiting the RunTime.

WARNING -- Exiting the MS-Windows RunTime abnormally can lead to data corruption.

4.9 Using the Standard NPL Runtime

The standard NPL RunTime is also included with the MS-Windows RunTime Package. Niakwa recommends using only the MS-Windows version of the RunTime on any system operating with MS-Windows. However, there may be situations when the use of the standard version is necessary. For example, users of a Novell NetWare network who do not have access to MS-Windows, running under MS-DOS only, must use the standard RunTime version.

NOTE: For instructions on using the standard RunTime, please refer to the MS-DOS Supplement for more details.

4.9.1 Serial Number

The serial number used by the standard NPL RunTime is the same as that used by the MS-Windows RunTime. If the Gold Key security is installed on the hard drive, either RunTime can pass security from the hard drive, otherwise the RunTime prompts the user to insert the Gold Key diskette to pass security as discussed in Section 2.5 of the MS-DOS Supplement.

4.9.2 User Limit

If an application must have multiple tasks of the NPL RunTime executing, the developer should use the MS-Windows RunTime (using MS-Windows, as many tasks as can fit into the system memory are allowed to execute simultaneously).

NOTE: Niakwa strongly recommends using the MS-Windows version of the RunTime, when using MS-Windows.

For the standard Novell NetWare RunTime, the user count is adjusted by one each time a regular RunTime is executed.

4.9.3 Device Sharing

Device sharing, particularly non-network files and devices, is not allowed under the standard version of the Niakwa RunTime. Attempts to use the standard RunTime under MS-Windows should be avoided, particularly when the MS-Windows RunTime or another standard RunTime window is being used.

WARNING -- Data corruption can result if the standard RunTime and a MS-Windows RunTime concurrently access the same local file.

NOTE: The MS-Windows RunTime fully supports device sharing. Refer to Chapter 6 for more information on device sharing with the MS-Windows RunTime.

CHAPTER 5

DEVICE SUPPORT

5.1 Overview

This chapter discusses the various devices supported by the NPL under MS-Windows.

Section 5.2 discusses the handling of storage devices (diskimages and diskettes) with the MS-Windows RunTime.

Section 5.3 discusses on-line printing with the MS-Windows RunTime.

Section 5.4 discusses using the MS-Windows Print Manager for printing with the MS-Windows RunTime.

Section 5.5 discusses using serial devices with the MS-Windows RunTime.

Section 5.6 discusses monitor support with the MS-Windows RunTime.

Section 5.7 discusses keyboard support with the MS-Windows RunTime.

Section 5.8 discusses mouse support with the MS-Windows RunTime.

5.2 Storage Devices

The following sections discuss how the MS-Windows RunTime addresses NPL storage devices (i.e., diskimages and diskettes).

5.2.1 Diskimages

"Hogging" diskimage devices is fully supported with the MS-Windows RunTime. Refer to Chapter 5 of the MS-DOS Supplement for additional information.

NOTE: The MS-DOS SHARE program must be loaded for this to operate correctly with local files.

Files not "hogged" are released back to the system during a KEYIN execution to make the file handles available to other windows. These files are reopened automatically on demand. Consequently, "hogged" diskimages can generate several additional open files that may require an increase in the open files parameter specified in the CONFIG.SYS file. Refer to Section 2.3.1 more information.

$OPEN Under Novell NetWare

Under Novell NetWare, if a $OPEN is attempted on a diskimage that is hogged by another workstation, all tasks on the workstation (NIAKWA and non-NIAKWA) slow down considerably. This is because all lock requests that are "wait for it" types (without line number exit) use a NetWare call which requests the lock with a time-out at the server.

This time-out is intended to keep the network from filling up with traffic in cases where there is temporary file contention. Only one network request every five seconds goes to the file server, and returns immediately if the lock is released.

While waiting for the results of the lock request, MS-Windows cannot service any other DOS requests or MS-Windows messages. This results in extremely slow performances and response in other windows (while the lock request is pending).

To allow the developer to fine tune this delay and improve performance, two RTI-WIN.INI options, **LockWaitTimeOut** and **LockRetryDelay**, are available to control the behavior of these lock requests.

Refer to Section 3.4 for details on the use of these options.

NOTE: No single value is likely to work well in all environments. If the time-out and retry delays are both short, the network will fill up with lock request traffic whenever file contention occurs. If the time-out delay is too long, there is a risk of waiting an un-necessarily long time for a locked file after it had been released or of missing the chance of locking the file to another user's workstation.

Exclusive Use of The Windows Resources While a Diskimage is Locked

Under Novell NetWare, if a non-Niakwa application that requires exclusive use of the Windows resource starts while a Niakwa task has a diskimage locked (using $OPEN), the diskimage remains locked until the non-Niakwa application completes the task and the Niakwa task is reactivated. During this period, other nodes that attempt to access the locked diskimage are forced to wait, giving the appearance that the entire network is very slow.

To avoid this odd timing problem, a $OPTIONS byte is available that indicates that the RunTime should not yield to other MS-Windows tasks while $OPEN's are active. The "no-yield" behavior can be implemented by use of byte 43 in $OPTIONS.

Byte 43 of $OPTIONS is implemented as follows:

HEX(00) [default] - yield to other Windows tasks even if $OPEN's are outstanding.

HEX(01) yield to other Windows tasks only if no $OPENs to network files have been granted.

Refer to Chapter 7 for details on $OPTIONS settings under MS-Windows.

NOTE: A RTIWIN.INI option is also available to avoid this timing problem. This is the ExclusiveWhenNetworkLocksHeld option. Refer to Section 3.4 for details on this option.

The application can override the ExclusiveWhenNetworkLocksHeld option by setting byte 43 in $OPTIONs to HEX(01).

Use of $OPTIONS is recommended for situations where the developer wishes to avoid modification of the RTIWIN.INI file. This can also be used with specific parts of the application (i.e., one that is likely run in the background or take long enough that the control is most likely be passed to other window applications). The ExclusiveWhenNetworkLocksHeld RTIWIN.INI option is intended for situations where the developer does not want to make modifications to the original software.

NOTE: The implementation of the Byte 43 in $OPTIONS and/or the ExclusiveWhenNetworkLocksHeld option in RTIWIN.INI could cause uneven operation of other open Windows (they could effectively stop running while $OPEN's are active).

Due to the specifics of Niakwa's MS-Windows implementation, the "no-yield" behavior does not apply when waiting for keyboard input. While waiting for keyboard input, the RunTime releases locked files and yields control to other MS-Windows tasks.

5.2.2 Diskettes

Raw diskette support is provided as documented in Chapter 6 of the MS-DOS NPL Supplement with the exceptions noted below.

The requirements for DOS base memory under MS-Windows for raw diskette buffering is 32K under Release IV. This memory is only required when accessing raw media. If the required base memory is not available, the statement attempting to access the raw diskette fails with a NPL P48 (Illegal Device Specification), with an MS-DOS error code of two (2), (file not found).

WARNING--"Hogging" of raw diskettes excludes access to the diskette from other Niakwa RunTime tasks only. This does not prevent other programs running concurrently under MS-Windows from accessing the diskette.

5.3 On-line Printing

To send output to a printer without the use of the MS-Windows Print Manager, the NPL device statements should use the normal NPL naming conventions. Refer to Chapter 5 of the NPL MS-DOS Supplement.

Several differences exist on how printing to on-line devices functions under the MS-Windows RunTime. These differences are discussed in the following sections.

5.3.1 Printing to Local Devices

The following section discusses printing to local print devices under the MS-Windows RunTime.

Use of $OPEN

Printer sharing using $OPEN/$CLOSE is fully supported between NPL tasks on the same workstation. Use of $OPEN/$CLOSE is requried to prevent intermingled output.

Under the NPL MS-Windows RunTime, PRINT statements used with $OPEN to a local printer may be slower than when no $OPEN is used. This is due to a 200ms delay produced by the RunTime if the parallel port output is refused (due to the internal buffer being full). This delay assumes that the buffer, approximately 512K, needs to be flushed.

Due to this delay and the fact that the MS-Windows print drivers expect large blocks of data, but can only keep track of a small number of blocks of any size, excessive delays may appear for PRINT statements with few or no characters. To address these issues two RTIWIN.INI options, **ParallelFullDelay** and **ParallelRetryCount** are available. Refer to Section 3.4 for details on these options.

5.3.2 Printing under Novell NetWare using LPTx Devices

The following section discusses printing under NPL for MS-Windows on a Novell NetWare system.

The Novell NetWare CAPTURE command redirects output from the designated LPTx device to the Novell NetWare spooler as under the standard Novell NetWare version of NPL. If several RunTimes tasks are running concurrently, and each sending output to a LPTx device, the Novell NetWare spooler creates a different spool file for each task. Thus, each tasks output is printed separately

5.4 Using the MS-Windows Print Manager

Use of the MS-Windows Print Manager is supported by the Niakwa MS-Windows Run-Time. This support allows for many variations as described below.

NOTE: This section assumes an understanding of the MS-Windows Print Manager. Refer to the MS-Windows documentation for more information.

The syntax used in the $DEVICE statement for accessing the MS-Windows Print Manager is as follows:

```
$DEVICE(/215)=">(printer file specification)DocName XXX=X"
```

Each of the individual parameters (printer file specification, DocName, and other options) are described in detail below.

NOTE: Output directed to the MS-Windows Print Manager may itself be redirected to a Novell NetWare Spooler. This operates transparently to the NPL application.

5.4.1 Printer File Specification

The "printer file specification" can be set to one of the following options:

1. To use the MS-Windows default printer device.

   A null string specifies that the [windows]/device= specification in the MS-Windows WIN.INI file is to be used.

   For example:

   ```
   ">()"
   ```

2. To use one of the predefined MS-Windows print devices.

   The following format should be used as described in the MS-Windows documentation for MS-Windows print device specifications.

   DeviceName, DriverName, Output

   For example:

   ```
   ">(HP ThinkJet (2225 C-D)THINKJET,LPT1:)"
   ```

3. To select a RunTime configured printer device.

   This option uses the print device specified in the Niakwa RTIWIN.INI file by PrinterConfig and the @partialkey keyword. Refer to Section 3.4 for details.

   For example:

   ```
   (@partialkey)
   ```

4. To display a dialog box for operator-assisted printer selection.

   This option asks the operator, from a dialog box, which MS-Windows print device is to be used at the first attempt to access the device. The entries that are displayed are the descriptive driver names for the predefined MS-Windows print devices are maintained by the Printers Option of the MS-Windows Control Panel.

   The format of this printer file specification is as follows:

   For example:

   ```
   >(?initialselect)
   ```

   where the initialselect string may be one of the following options:

   ```
   Null              The MS-Windows default printer is selected.
   @partialkey       The RTIWIN.INI is looked at for the actual selectstring.
   Other             The string specified is used to select a printer if one matches.
   ```

   The actual value of the select string is used to decide which printer entry is initially highlighted.

NOTE: If the Dialog Box is canceled without selecting a printer, a P48 (Illegal Device Specification) error is generated.

For example:

```
">(?Generic)"
```

It is also possible to combine the above syntax for the printer file specification as in the examples shown below:

Example 1:

```
$DEVICE(/204) = ">(@1)"
```

and the RTIWIN.INI file(s) have

```
PrinterConfig1=?
```

is equivalent to

```
$DEVICE(/204) = ">(?)"
```

Here the dialog box appears and the default printer is highlighted.

Example 2:

```
$DEVICE(/204) = ">(?@1)"
```

and the RTIWIN.INI file(s) have

```
PrinterConfig1=Generic
```

is equivalent to

```
$DEVICE(/204) = ">(?Generic)"
```

Here the dialog box appears and default printer starting with Generic (if any) is highlighted.

NOTE: MS-Windows printer specifications must be enclosed within matched parentheses.

5.4.2 DocName

The DocName must appear immediately after the ending parenthesis for the printer file specification and may not contain any spaces.

Documents in the MS-Windows Print Spooler can be identified in the Print Manager display by looking for the name (DocName) that immediately follows the ending parenthesis in the $DEVICE statement. If no document name is specified, the document name submitted to the MS-Windows Print Spooler is of the form:

```
"Caption Partition#"
```

where caption is the current window's caption.

5.4.3 Other Options

Other options are available with the MS-Windows Print Manager as described below:

Using the MS-Windows Printer Driver Configuration Box

The MS-Windows print driver configuration box can be invoked by using the CFG= Y/N parameter with the $DEVICE statement.

For example:

```
$DEVICE(/215)=">() CFG=Y"
```

If this option is set to Y, the first attempt to use the printer after assigning $DEVICE invokes the MS-Windows printer configuration dialog box. If the MS-Windows printer selection dialog box appears, the status can be checked, modified, or canceled.

Locking in a Printer Specification

A printer specification also can be set by using the MS-Windows SET= Y/N (default N) option. If this option is set to Y, any printer selection that is used successfully sets the $DEVICE value to the option selected, replacing any specification that is shown.

For example:

```
">(?) SET=Y"
```

At the first print attempt, the MS-Windows printer select dialog box appears, with the default printer highlighted. If an item such as "HP ThinkJet (2225 C-D),THINKJET,LPT1:" was selected and successfully opened, the $DEVICE would be set to:

```
">(HP ThinkJet (2225 C-D),THINKJET,LPT1:) SET=Y"
```

Effectively, this "locks in" the printer unless the $DEVICE is reset by the application.

NOTE: The value inside the parentheses is replaced, the options outside the parentheses are retained. If the total length exceeds the maximum for a legal $DEVICE (50 characters), no replacement occurs.

5.4.4 Using Control Codes with the MS-Windows Print Spooler

All control codes in the range HEX(00) to HEX(1F) (after optional translation due to the Niakwa XLA= Y option) that are sent to the MS-Windows Print Spooler are interpreted by the MS-Windows RunTime and removed from the data stream.

NOTE: The MS-Windows Spooler is different from the XENIX/UNIX spooler and the Novell NetWare spooler which can accept arbitrary data and control codes.

The MS-Windows RunTime interprets the following control codes:

- HEX(0D) Go to start of line.
- HEX (0A)Advance to next line. If off page limits, go to new page.
- HEX(0C) Form feed. Multiple HEX(0C) at the top of the page are ignored.

Closing the printer device by $CLOSE or $DEVICE= < any value > completes the page and ends the spool stream.

If ERR= Y is specified in the $DEVICE specification, an error may occur only at a page break.

An ALF= N specification in the $DEVICE line may allow overprinting on the same line depending on the printer driver.

NOTE: Make sure the printer settings, the applications settings, and the MS-Windows Print Manager settings are the same for the number of lines per page. If the application is tracking printer setup parameters and the values are different from the MS-Windows settings, the number of lines printed per page may vary significantly.

5.4.5 Using the MS-Windows Print Manager Under Novell NetWare

Applications that can accept the limitations of the MS-Windows Print Manager (the primary limitation is that printer control sequences other than HEX(0D), HEX(0A), and HEX(0C) cannot be used when directing output to the Print Manager), can access the Novell NetWare spooler using the MS-Window Print Manager (Print Spooler).

NOTE: Output directed to the MS-Windows Printer Manager may itself be redirected to a Novell NetWare spooler. This operates transparently to the NPL application.

5.5 Serial Ports

Serial as communications is supported by the MS-Windows RunTime. Serial communications as described in Section 5.7 of the NPL MS-DOS Supplement is supported. In addition, several options in the RTIWIN.INI file can be used to configure the serial ports for communication. For details refer to the options: **CommInputBufferSize, CommOutputBufferSize**, and **CommFlushDelay** in Section 3.4.

5.5.1 Sharing Serial Ports

Sharing of serial devices is not supported. If an attempt is made to access an existing device while it is already in use by another RunTime window or by another MS-Windows task (such as terminal emulation or the MS-Windows Print Spooler), the RunTime HELP Screen appears with the message "xxx is busy" and the other application must release the device before processing can continue.

5.6 Monitor Support

The Niakwa RunTime operates in graphics mode under MS-Windows. Monitor characteristics, except as noted in this section, are identical with characteristics described in Chapter 6 of the appropriate NPL Supplement for /G mode.

This section discusses the MS-Windows RunTime support of different fonts and 132 column screen mode.

5.6.1 Fonts

The following sections discuss the different fonts and font options available to the MS-Windows RunTime.

Available Font Files

The MS-Windows RunTime Package includes two font files, BASFONTS.FON and IBMFONTS.FON. Both font files contain a series of fonts that have been specially designed to work with the NPL MS-Windows RunTime.

The BASFONTS.FON font file contains the standard NPL character set. Use of this font allows the display of all standard NPL characters while using the MS-Windows RunTime.

The IBMFONTS.FON font file contains the standard IBM PC character set. Use of this font allows the display of all standard IBM PC characters while using the MS-Windows RunTime.

NOTE: The BASFONTS.FON font file is the default font file used by the MS-Windows RunTime. Use of this font file requires no additions to the RTIWIN.INI file.

To allow the MS-Windows RunTime to use the IBMFONT.FON font file, it is necessary to add the following lines to the [GENERAL] or application section of the the system or local RTIWIN.INI file:

```
FontCharSet = 179
FontFaceName = IBASIC
```

Refer to Section 3.4 for more information on the RTIWIN.INI file.

Adding a Font File to MS-Windows

To use either font file, it is necessary to add the font to the MS-Windows fonts in the MS-Windows Control Panel. This is accomplished by the steps shown below.

NOTE: This example assumes the use of the BASFONTS.FON font file. If the IBMFONTS.FON font file is needed, substitute that file for BASFONTS.FON.

1. Make sure the file BASFONTS.FON is located in the \WINDOWS directory.

2. Select the MS-Windows Program Manager.

3. From the Program Manager, select the Control Panel option.

4. From the MS-Windows Control Panel, select the Fonts option.

5. From the Font option, select the Add option. This displays a list of the available *.FON files in the \WINDOWS directory.

6. Select the BASFONTS.FON file.

7. Select OK.

This procedure adds the Niakwa fonts to the installed MS-Windows fonts.

WARNING--Niakwa recommends the use of the BASFONTS.FON font file. Use of other font files do not allow the entire NPL character set to be available to the RunTime.

Several options in the RTIWIN.INI file can be used to set the default font when a Run-Time task starts. These options are: **FontFaceName,** and **FontCharSet**. For more information on these options, refer to Section 3.4.

True Type Fonts

Support for MS-Windows True Type fonts is limited to those which have a fixed pitch (non-proportional fonts). When fixed pitch True Type fonts are used, the MS-Windows RunTime ensures that the spacing of characters on the screen is correct (if the correct RTIWIN.INI options for selecting alternate fonts have been used).

The only fixed pitch True Type font that ships with MS-Windows is called "Courier New", which has an ANSI character set. To select this font, enter the following in the [GENERAL] or application section of the RTIWIN.INI file:

```
FontFaceName=Courier New
FontCharSet=0
```

NOTE: True Type fonts do not use the standard NPL character set, consequently not all NPL characters may not display when using this font (the $SCREEN table selects the closest ANSI value to the standard NPL character set).

Dynamic Resizing

The size of the current font used can be automatically resized if the size of the current window is changed. Dynamic resizing is accomplished through the Autosize feature of the MS-Windows RunTime. Refer to Section 4.4 for more information on this interactive option.

NOTE: The last size and type of font used before the RunTime task is closed is stored in the RTIWIN.INI file.

Modifying the Niakwa Fonts

Source files for the Niakwa fonts are provided with the NPL MS-Windows Development Package and can be modified by the developer. The MS-Windows Supplementary Files Diskette contains a series of files in the \BFONTS and \IFONTS directories, which contain all files necessary to modify the Niakwa fonts using the MS-Windows Software Developers Kit.

NOTE: Information on revising the Niakwa-supplied font files is provided in Appendix B.

HINT: If it is necessary to modify the screen translation table with the Niakwa Utilities, the developer should name the modified file SCREEN.WIN. The MS-Windows RunTime always looks for this file before using the internal screen table built into the MS-Windows RunTime.

Screen translation values are ignored unless the selected FontCharSet is not an NPL character set (178).

5.6.2 132-Column Support

Support for 132-column mode is supplied with the MS-Windows RunTime. This feature is supported as documented in Section 7.4.12 of the NPL Programmer's Guide.

5.7 Keyboard Characteristics

The following are the default keyboard mappings for the MS-Windows RunTime. In most cases, these keyboard mappings are the same as the MS-DOS version of NPL except:

1. All keyboard combinations using ALT or F10 are reserved by MS-Windows and consequently not used. Special function keys '9-'15 and shifted versions of these keys are remapped from the MS-DOS conventions.

2. Some combinations of keystrokes that generate the same keystrokes under MS-DOS can be distinguished under MS-Windows (i.e., SHIFT-ARROW keys). The MS-DOS key combinations are also supported (except where this is not possible due to item 1).

Commonly used key differences:

HALT	CTRL-BREAK

Editing key differences:

SHIFT-CTRL-F3 or SHIFT-INS	Inserts a line feed within a line of text.
SHIFT-CTRL-F10	Deletes characters from the current cursor position to the end of line.

Edit Mode SF key differences:

CTRL-F2	Deletes one character at the current cursor position.

MS-Windows Default Keyboard Equivalences Table:		
Niakwa Code	**Niakwa Virtual Key**	**MS-Windows Key**
08	BACKSPACE	BACKSPACE
0D	RETURN	ENTER
81	CLEAR	?
82	EXECUTE	HOME
83	CONTINUE	?
A1	LOAD	CTRL-7/HOME CTRL-ENTER
E5	SHIFT-ERASE	CTRL-W
FF'A0'xx	UNDERSCORE (DEAD KEY)	?
'00-'08	SF '0...'8	F1...F9
'08-'0F	SF '8...'15	CTRL-F1...F9
'10-'18	SHIFT SF '0...'8	SHIFT F1...F9
'18-'1F	SHIFT SF '8...'15	SHIFT-CTRL-F1...CTRL-F8
'42	PREV-SCREEN	9/PG UP
'43	NEXT-SCREEN	3/PG DN
'45	SOUTH	2/SOUTH
'46	NORTH	8/NORTH
'49	DELETE	./DEL
'4A	INSERT	INS

MS-Windows Default Keyboard Equivalences Table:		
Niakwa Code	**Niakwa Virtual Key**	**MS-Windows Key**
’4C	EAST	6/EAST
’4D	WEST	4/WEST
’4F	RECALL	CTRL-R
’50	SHIFT-CANCEL	CTRL-K SHIFT-1/END
’52	SHIFT-PREV-SCREEN	CTRL-P CTRL-9/PG UP SHIFT-9/PG UP
’53	SHIFT-NEXT-SCREEN	CTRL-N CTRL-2/PG DN SHIFT-3/PG DN
’55	SHIFT-SOUTH	SHIFT-2/SOUTH CTRL-2/SOUTH
’56	SHIFT-NORTH	SHIFT-8/NORTH CTRL-8/NORTH
’59	SHIFT-DEL	SHIFT-./DEL CTRL-./DEL
’59	SHIFT-INS	SHIFT-0/INS CTRL-0/INS
’5C	SHIFT-EAST	CTRL-4/EAST SHIFT-4/EAST
’5D	SHIFT-WEST	CTRL-6/WEST SHIFT-6/WEST
’5F	DEC TAB	CTRL-T
’7C	GL	CTRL-G
’7D	SHIFT-GL	CTRL-Z
’7E	TAB	TAB
’7F	SHIFT-TAB	SHIFT-TAB
’E1	HELP	ESC
’F0	EDIT	1/END
HALT	HALT	CTRL-BREAK

NOTE: The SF'8 and SF'24 keys appearing twice in the above table is correct as they can be entered two ways.

If it is necessary to modify the keyboard translation table with the Niakwa Utilities, the developer should name the modified file KEYBOARD.WIN. The RunTime always looks for this file before using the internal keyboard table built into the MS-Windows RunTime.

WARNING--If the keyboard is to be remapped under MS-Windows, the simple and complex codes generated under MS-Windows differ substantially from the same values generated under MS-DOS. Consequently, direct modifications to the $KEYBOARD system variable that work under MS-DOS do not function correctly under MS-Windows. Most combinations of keys with SHIFT, CTRL, or both SHIFT and CTRL generate distinct complex codes under the MS-Windows version.

5.8 Mouse Support

The following section details the support for the mouse interface with the MS-Windows RunTime. Refer to Section 4.4 for details on the support of the Browse and SF Keys with the mouse.

The location of the mouse is reported to $MACHINE each time a mouse button is pressed. Mouse events are reported to the NPL application by generation of special function key codes as follows:

'F1 - left button pressed
'F2 - left button released
'F3 - left button pressed (within double click time)
'F4 - right button pressed
'F5 - right button released
'F6 - right button pressed (within double click time)
'F7 - dragged north (up)
'F8 - dragged south (down)
'F9 - dragged east (right)
'FA - dragged west (left)

When a key is read by the NPL KEYIN statement, bytes 23 and 24 of $MACHINE contain the Y (row) and X (column) address of the current location of the mouse when the event occurred, if it is "on screen". When the cursor is outside the screen area (or if there is no mouse) these bytes contain high values (HEX(FF)).

NOTE: Several utilities included in UTILITY.BS2 have been modified to support the mouse using the above settings.

There are two RTIWIN.INI options that can control the key codes generated by the mouse, **MouseClickKeys** and **MouseDragNSEWKeys**. Refer to Section 3.4 for more information on these options.

For example:

```
MouseClickKeys=0,0,82,0,0,82
MouseDragNSEWKeys='46,'45,'4C,'4D
```

changes the mouse function so that dragging the mouse is equivalent to cursor keys, the press and release don't generate any keys, and either the right or left mouse button double-clicked generates an EXECUTE. This setup may be used for applications that cannot accommodate the use of the mouse keys or can't be modified to recognize the mouse keys.

It is recommended that applications that support the mouse assume the standard keys are generated.

CHAPTER 6

MULTI-USER CAPABILITIES

6.1 Overview

Multi-user capabilities are supported with the Niakwa MS-Windows RunTime. This chapter details the specific features of this support.

Section 6.2 discusses unique terminal identification and provides an example of creating unique network-wide partition values.

Section 6.3 discusses device sharing.

Section 6.4 discusses intertask communications.

6.2 Unique Terminal Identification

The following section discusses the methods available with the NPL MS-Windows Run-Time for unique terminal identification.

6.2.1 General Principles

Configuration of unique terminal identification is quite flexible with the MS-Windows RunTime. By default, the MS-Windows RunTime operates as follows:

- #ID is always zero on MS-DOS versions. On Novell NetWare network installations, #ID is generated as described in Chapter 5 of the Novell Addendum.
- $NETID works as documented in Chapter 5 of the Novell Addendum.
- #TERM on MS-DOS is always one. On Novell NetWare network installations, #TERM values are generated as described in Chapter 5 of the Novell Addendum.
- #PART is generated sequentially for each task, in order of execution of the task, on each workstation. The first task executed is assigned a #PART of 1, the second is assigned a value of 2, and so on.

Several options can be set in the RTIWIN.INI file to modify this default behavior:

- The **Partition** option can be used to assign a specific #PART value to a specific task.
- The **ReservedPartitions** option can be used to reserve one or more #PART values.
- The **NetworkPartitionsFile** option can be used to remap initial #PART values to network-wide unique #PART values.
- The **TerminalIsPartition** option can be used to remap #TERM to be equal to #PART.

Refer to Section 3.4 for further details on these options.

The following guidelines may be useful for determining how best to use these options for Niakwa MS-Windows applications:

- Applications that require the same #PART value to be generated each time the application is executed should use the **Partition** option. If it is desirable to allow the user to execute multiple copies of the application with a consistent #PART value for each copy, then it is necessary to make copies of the BOOT program under different names, set up separate Program Menu icon entries for each BOOT program, and assign specific #PART values for each boot program in RTI-WIN.INI (or NetworkIniFile).

- On Novell NetWare network installations where some users are using the MS-Windows RunTime and some are using the standard MS-DOS or Novell NetWare RunTime, set **ReservedPartitions = 1** in the NetworkIniFile. This ensures that all MS-Windows tasks have #PART values of 2 or greater.

- Applications that internally use #ID and #PART (whether or not #TERM is also used) to generate unique terminal identification need do nothing further.

- Applications that use #ID and #TERM to generate unique terminal identification should use the **TerminalIsPartition** option.

- Applications that use #PART and #TERM to generate unique terminal identification but do not use #ID, should use a TERMINAL.TBL file.

- Applications that use only #PART to generate unique terminal identification must use a TERMINAL.TBL file and **NetworkPartitionsFile**.

- Applications that use #TERM only to generate unique terminal identification must use a TERMINAL.TBL file, **NetworkPartitionsFile**, and must specify **TerminalIsPartition**.

6.2.2 Creating Unique Network Partition Values

The following section provides an example of the steps necessary to implement and set up unique network-wide #PART values.

If the use of #PART is the only factor available to determine unique tasks, then the steps shown below should be followed. Refer to Section 3.4 and 6.2 for more information.

NOTE: Niakwa recommends that, whenever possible, an application use a combination of #ID, #TERM and #PART to determine unique tasks.

Three files must be created or modified to allow for unique network-wide #PART values. These files are:

RTIWIN.INI	Local to each workstation using the network version of MS-Windows.
NetworkIniFile	F:\BASIC2C\RTIWIN.INI in this example
NetworkPartitionsFile	F:\BASIC2C\NETPARTS.TBL in this example

The examples below describe modifying or creating the above files. Editing can be performed with any text editor (i.e., MS-Windows Notepad, EDIT, EDLIN, etc.).

1. Modify the GENERAL section of the NetworkIniFile (F:\BASIC2C\RTIWIN.INI) to appear as:

   ```
   [GENERAL]
   ReservedPartition= 1
   TerminalisPartition= 1
   NetworkPartitionsFile= F:\BASIC2C\NETPARTS.TBL
   ```

   where:

   ReservedPartition= 1 reserves the first partition value so that unique network-wide partition numbers are generated using either the MS-Windows RunTime or the standard Novell NetWare RunTime (this will always have a partition number equal to 1).

   TerminalIsPartition= 1 sets the #TERM values equal to the #PART values (if desired).

   NetworkPartitionsFile= F:\BASIC2C\NETPARTS.TBL sets the NetworkPartitionsFiles, from which the #PART values are determined to the NETPARTS.TBL file on drive F on the BASIC2C directory on the host file server.

2. Create a separate BOOT program for each task to appear on the MS-Windows Program Manager. This can be a copy of the original BOOT program under a slightly different name.

3. Create an entry in the user's local RTIWIN.INI file for each task. Use the BOOT program names as defined by the tasks in step 2 and specify the Partition option to assign a unique initial partition number.

NOTE: Steps 2 and 3 are necessary to assure that the initial #PART value generated for each task to be executed on a workstation is unique and can, therefore, be mapped to a unique entry in the NetworkPartitionsFile.

For example, if a boot file is named \BASIC2C\MYAPP\BOOT1.OBJ and its initial unmapped #PART value should be 2, the following would be located in the user's local RTIWIN.INI file:

```
[F:\BASIC2C\MYAPP\BOOT1.OBJ]
Partition= 2
```

NOTE: Any MS-Windows RunTime task that will be executed on a particular workstation must be set up in the user's local RTIWIN.INI file with a Partition option value. If an attempt is made to execute a task that does not have a ReservedPartition value assigned to it, two different error messages may be generated depending on the remapped partition values of the tasks already executing.

One error message that may be generated is caused when the same initial #PART value (the value before remapping) is generated as a previous partition (assuming that the remapped #PART value is different from the initial value). This causes the following error message to be displayed:

Partition X mapped to Network partition Y already used,
cannot start this application.

One way to prevent this error is to make sure that every boot file specified in the RTIWIN.INI file uses the Partition option and has a unique value.

The other error message that may be generated is caused when no previous remapping of the new partition has already occurred. If this is the case, then the following error message displays:

Terminal X partition Y not configured in NetworkPartititonsFile < filename>.

Once either of the above error messages is cleared, the following error message also displays:

Cannot determine Partition number.

4. Create the NetworkPartitonsFile (F:\BASIC2C\NETPARTS.TBL).

   This file, when used with TERMINAL.TBL (which generates unique #TERM/#PART values for each node on a network), allows the MS-Windows RunTime to generate unique #PART values for each MS-Windows RunTime task running from workstations on a network-wide basis.

   The MS-Windows RunTime assigns a new #PART value by searching for an entry containing the original #TERM value and the original #PART value (as generated by use of the **Partition** option for each task in the user's local RTI-WIN.INI file). The new #PART value assigned is the number of the matching entry in the **NetworkPartitionsFile.** For example, a task matching entry 7 is assigned a #PART value of 7; a task matching entry 8 is assigned a #PART of 8.

   To ensure unique #PART generation for installations where some operators are using the MS-Windows version and some workstations are using the standard Novell NetWare RunTime (this is normally the case and is assumed in this example), two steps are necessary:

   a. Set the **ReservedPartitions= 1** option in either the **NetworkIniFile** or the user's local RTIWIN.INI file.

b. At the start of the **NetworkPartitionsFile**, place an entry for each user, specifying the #TERM value of the workstation and a #PART value of 1. This reserves the first X #PART values for the DOS tasks. The first X values are then reserved, so that network-wide partition numbers generated using the MS-Windows RunTime are never identical to those generated using the standard Novell NetWare RunTime. This allows users to switch between the MS-Windows RunTime or the standard Novell NetWare RunTime as required by the application. The #PART and #TERM values reserved are the same as the #PART and #TERM values generated by TERMINAL.TBL.

6.2.3 Step-By-Step Example

The following is a step by step example using sample #ID, #TERM, and #PART values. This example assumes that only three workstations exist on the example network.

In this example, the TERMINAL.TBL file contains the following values:

123
148
167

Based on the above values, the following are the #TERM values generated by the MS-Windows RunTime for the #ID values in the TERMINAL.TBL file.

#ID	**#TERM (from TERMINAL.TBL)**
123	1
148	2
167	3

NOTE: The NPL variable, $NETID, can be used in conjunction with the NETID.TBL file to determine the #ID values used initially by the RunTime. Refer to Chapter 5 of the NPL Novell Netware Addendum for details.

Since this example assumes only three workstations on the network, the first three entries in the **NetworkPartitionsFile** specify a #PART value of 1. This reserves the #TERM/#PART values of 1, 2, and 3 for use by the standard Novell NetWare RunTime (refer to the first three entries in the example below).

For our example programs, the number of partitions necessary on each workstation running the MS-Windows RunTime is as follows:

Workstation	**#TERM value**	**Max. Number of Windows Tasks for This Workstation**
Workstation A	1	3
Workstation B	2	2
Workstation C	3	4

The NetworkPartitionsFile for this example would then contain the following:

```
1 1
2 1
3 1
1 2
1 3
1 4
2 2
2 3
3 2
3 3
3 4
3 5
```

NOTE: The above example would generate the following #PART values for each workstation:

#TERM	Original #PART Value Assigned for Each Workstation	New #PART Value from NetworkPartitionsFile
1	1	1*
2	1	2*
3	1	3*
1	2	4
1	3	5
1	4	6
2	2	7
2	3	8
3	2	9
3	3	10
3	4	11
3	5	12

* These partition entries are reserved for the standard DOS RunTime when used in conjunction with the MS-Windows RunTime in a Novell NetWare environment.

The values shown for each task remain constant (i.e., the task run on the workstation with #TERM= 2 with a Partition value of 3 always generates a unique #PART value of 8 for that task).

NOTE: The first three #PART values are reserved for the standard RunTime Tasks. #PART is set equal to #TERM for the standard RunTime since TERMINAL.TBL is in use.

To understand the above example, notice that workstation C (#TERM= 3) can have only four RunTime tasks active at any one time and that the fourth RunTime task (BOOT4.OBJ) on workstation C is always assigned #PART= 12. The local RTIWIN.INI file for workstation C would contain the following entries:

NOTE: The entries in boldface apply directly to the discussions above. The other entries are for illustration only.

```
[GENERAL]
NetworkIniFile = F:\BASIC2C\RTIWIN.INI
ReservedPartitions = 1
NetworkPartitionsFile = F:\BASIC2C\NETPARTS.TBL
TerminalsPartition = 1

[F:\BASIC2C\MYAPP\BOOT1.OBJ]
Partition = 2
Window = 22 87 594 479 0
SFKeys = 280 150 0
AutoSize = 1 7 13 80

[F:\BASIC2C\MYAPP\BOOT2.OBJ]
Partition = 3
Window = 122 187 294 379 0
SFKeys = 280 150 0

[F:\BASIC2C\MYAPP\BOOT3.OBJ]
Partition = 4
Window = 210 47 394 279 0
AutotSize = 1 7 13 80

[F:\BASIC2C\MYAPP\BOOT4.OBJ]
Partition = 5
Window = 22 87 594 479 0
```

where BOOT1.OBJ - BOOT4.OBJ are identical boot programs.

Based on the above example, the following are the mapped network-wide partition values that would be generated by the RunTime:

BOOT File Name	Original #PART Value	Mapped #PART Value
BOOT1.OBJ	2	9
BOOT2.OBJ	3	10
BOOT3.OBJ	4	11
BOOT4.OBJ	5	12

6.3 Device Sharing

Device sharing is permitted using the MS-Windows RunTime. This includes the availability of $OPEN and $CLOSE to allow one partition to take control (hog) a device for its own use, while the other partition waits. Refer to Chapter 5 for more information.

6.4 Intertask Communications

This section describes the NPL $PSTAT and $MSG statements and how they can be used for intertask communications with the MS-Windows RunTime.

6.4.1 $PSTAT

$PSTAT can be used to pass information between MS-Windows RunTime partitions on the same PC. Refer to the NPL Statements Guide under $PSTAT for details on using the syntax of this statement.

NOTE: Each workstation on a Novell NetWare network has a different set of $PSTAT values.

6.4.2 $MSG

$MSG can be used to set the system message displayed for different tasks on the same workstation whenever a RESET or CLEAR is performed within a task. Refer to the NPL Statements Guide under $MSG for details on using the syntax of this statement.

NOTE: Each workstation on a Novell NetWare network has a separate $MSG value.

CHAPTER 7

PLATFORM-SPECIFIC LANGUAGE FEATURES

7.1 Overview

This chapter discusses the platform-specific language features for the MS-Windows Run-Time.

Section 7.2 discusses MS-Windows environment-specific statements.

Section 7.3 discusses background partition support under MS-Windows.

Section 7.4 discusses memory management under MS-Windows.

7.2 Environment-Specific Statements

The following section discusses the NPL environment-specific statements under MS-Windows

7.2.1 $MACHINE

The following are the MS-Windows specific $MACHINE values for the MS-Windows RunTime. All other $MACHINE values are the same as those for the standard NPL RunTime. Refer to Chapter 8 of the MS-Supplement for details.

Byte 1	RunTime Version "N" for MS-Windows	
Byte 3	Monitor Type "W" for MS-Windows	
Byte 4	Graphics Enabled "G" indicates that true box graphics are available.	
Byte 29	Indicates whether keyboard mouse events are supported	
	HEX(00)	Default, mouse not available
	HEX(01)	Occurs upon detection of an installed mouse

NOTE: $MACHINE is a 64-byte variable and must be treated as such or unpredictable results may occur. Refer to the Statements Guide, $MACHINE, for details on the exact syntax and use of this statement, as well as the contents of the remaining bytes of the variable.

7.2.2 $OPTIONS

The following are the MS-Windows specific $OPTIONS values for the MS-Windows RunTime. All other $OPTIONS values are the same as those for the standard NPL RunTime. Refer to Chapter 8 of the MS-Supplement for details.

Byte 33	The hex values under MS-Windows are the same as documented in the Statements Guide, but under MS-Windows the following new values exist.	
	HEX(10) bit - 0	Can Close RunTime tasks using the MS-Windows menu bar etc.
	HEX(10) bit - 1	Suppress the ability to Close RunTime tasks using the MS-Windows menu bar.
Byte 43	Specifies that the RunTime will not yield to another MS-Windows task while an $OPEN is active.	
	HEX(00) Default	Yield to other MS-Windows tasks even if $OPEN's are outstanding.
	HEX(01)	Yield to other MS-Windows tasks only if no $OPEN's to network files have been granted.

NOTE: $MACHINE is a 64-byte variable and must be treated as such or unpredictable results may occur. Refer to the Statements Guide, $MACHINE, for details on the exact syntax and use of this statement, as well as the contents of the remaining bytes of the variable.

7.2.3 $PSTAT

Refer to Section 6.4.1 or the NPL Statements Guide, $PSTAT for more information on $PSTAT.

7.2.4 $MSG

Refer to Section 6.4.1 or the NPL Statements Guide, $MSG for more information on $MSG.

7.2.5 $SHELL

The behavior of $SHELL under MS-Windows is quite different than under MS-DOS. Since MS-Windows is a true multi-tasking environment, $SHELL does not execute a sub-shell as it does under MS-DOS; it creates a new task. Refer to the NPL Statements Guide, $OPTIONS and Chapter 8 of the MS-DOS Supplement for more information on $OPTIONS.

Release IV Modifications to $SHELL

Under the original versions of the NPL for MS-Windows (Revision 3.20), program control returned back to the MS-Windows RunTime immediately after issuing the $SHELL command.

When a $SHELL command is issued under Release IV, the task or command issued by the $SHELL command is completed before returning control to the RunTime. For example:

```
$SHELL "COPY FILE1 FILE2"
```

$SHELL copies FILE1 to FILE2 and then return control back to the MS-Windows RunTime.

LAUNCH.EXE program

To allow the MS-Windows RunTime to operate as it did under Revision 3.20, in regards to the $SHELL command, a utility program named "LAUNCH.EXE" can be used.

For example:

```
$SHELL "LAUNCH COPY FILE1 FILE2"
```

$SHELL would start the LAUNCH program which in turn launches the DOS command COPY as a separate task. Launch then returns control to the RunTime without waiting for the launched program to complete. This COPY command is then completed as a separate task running concurrently.

Use of the $SHELL in this manner can cause potential negative side effects for the application since:

1. The application can not determine whether the launched task successfully completed.

2. Applications may attempt to access files that are expected to be produced by the launched task before they are available. If the launched task is a MS-DOS task that has files opened, attempting to access these files within the RunTime while the launched task is still active produces a SHARE error (the MS-DOS task has the files opened exclusively).

The above facts should be considered when designing applications that use LAUNCH.

The LAUNCH.EXE program is provided on the MS-Windows RunTime disks.

Starting Additional RunTime Tasks with $SHELL

$SHELL can also be used to start additional Niakwa RunTime and MS-Windows task (with the use of the LAUNCH.EXE command). The task is started and processed just as though the command line was entered from the MS-Windows Run command. Since there is no parent/child relationship between the originating task and the created task, closing or killing the originating task has no effect on the task created with $SHELL.

7.3 Background Partition Support

Background partitions are not supported under MS-Windows. $RELEASE TERMINAL performs no operation.

The MS-Windows environment does allow more than one RunTime task to be active at one time. The tasks can be run as MS-Windows minimized tasks as well. When a task is minimized it continues to execute, but any Browse of SF Key windows displayed are not visible as long as the task remains minimized. Refer to Section 4.5 for details.

HINT: Addition MS-Windows RunTime tasks can be spawned by using the LAUNCH program.

7.4 Memory Management

All NPL code and defined variables reside within a section of memory defined as the "user partition". With the MS-Windows RunTime, the size of the user partition is limited only by the memory available to MS-Windows.

Due to the dynamic nature of memory allocation in the MS-Windows environment, the MS-Windows RunTime allocates an initial 141K to the user partition. Unlike MS-DOS in which the maximum size of the user partition is reported, 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 memory available would leave insufficient memory for other tasks.

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, the MS-Windows RunTime 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 MS-Windows RunTime is unable to allocate the memory, the value of SPACEF then drops below 64K. To illustrate this, consider the following:

MS-Windows RunTime Environment	SPACEW Value	**SPACEF Value**
Initial RunTime Memory	139664	139456
Allocated 64K Variable	139664	73920
Allocate 8300 Bytes	139664	65600*
Allocate 20K Variable	205168	45080**

* Additional 64K segment allocated to user partition.

** Value SPACEF reports in the event the MS-Windows RunTime was unable to allocate the extra 64K.

CHAPTER 8

MIXED LANGUAGE PROGRAMMING

8.1 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 languages 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 MS-Windows. For a complete discussion on the general operation of mixed language programming, refer to Chapter 11 of the MS-DOS Supplement and Chapter 16 of the NPL Programmer's Guide.

The remainder of this Section continues to provide an overview of MS-Windows environment.

Section 8.2 discusses the contents of the MS-Windows BESDK.

Section 8.3 discusses the installation of the MS-Windows BESDK.

Section 8.4 discusses NPL external call support specific to the MS-Windows environment.

Section 8.5 discusses loading external libraries under MS-Windows

Section 8.6 discusses support of Microsoft C under MS-Windows.

Section 8.7 discusses support of Microsoft MASM Macro Assembler under MS-Windows.

Section 8.8 discusses shared data segments in DLL's.

Section 8.9 discusses custom resources in a DLL.

Section 8.discusses subclassing the main Niakwa Windows in a DLL.

Section 8.11 discusses flow control of external libraries

Section 8.12 discusses callbacks to NPL under MS-Windows

NOTE: The following chapter refers only to the MS-Windows BESDK package contained on the MS-Windows Supplementary Files Diskette. For information on the standard BESDK package, provided with the MS-DOS Development Package (on the BESDK diskette), refer to the MS-DOS Supplement.

Release IV features for external calls are is only available for the C programming language. Release III external call features in other languages (i.e. Pascal and Assembler) are upwardly compatible to Release IV, but no new Release IV features are supported for these languages.

8.1.1 Differences from MS-DOS/SuperDOS Releases

The MS-DOS and SuperDOS releases of the RunTime use the quick library mechanism of the Microsoft Linker, allowing the standard NPL program to load a specified set of routines after NPL starts. A similar approach is also used in the MS-Windows environment. However, instead of using the .QLB format for the quick library, the MS-Windows version uses the generally superior dynamic link library (DLL) format to contain the user-defined routines. The DLL code format is designed to be dynamically linked at execution time, and is used extensively elsewhere in the MS-Windows environment. One of the primary benefits of the DLL format is that MS-Windows automatically swaps unused portions of the DLL to disk, loading in portions as required.

8.1.2 Choosing the Development Environment

When coding and linking external routines, the following implications must be considered:

- To execute a MS-Windows executable, a running MS-Windows environment, version 3.1 or greater, must be present.

- Because the user subroutines must be linked into a dynamic link library, the Microsoft Linker (LINK), which is a part of the Microsoft development language of choice (C or MASM) must be present. In addition, the Microsoft MS-Windows Software Development Kit (SDK), which includes the resource compiler, include files, and start-up and support libraries designed for the MS-Windows environment, must also be present.

- Working with many examples is more efficient if the Microsoft MAKE or NMAKE utilities (or equivalent) are available. However, this is not a requirement to use BESDK.

NOTE: All provided makefile scripts work with either NMAKE or MAKE; however, when used with MAKE, several warning messages are displayed related to lines that are NMAKE pseudo-target instructions. These warnings can be ignored.

- To execute the instructions in the makefile, type "make makefile" or "nmake". For brevity, the rest of this chapter refers only to NMAKE.

8.1.3 Security

Any dynamic link libraries produced by using the BESDK procedures are not themselves physically copy-protected. Routines whose entry points are exported in the DLL are available to any MS-Windows programmer who knows the Library name, Entry point name and interface specification. If the routines contain proprietary or trade secret information, ensure that these sensitive routines do not operate unless enabled by some kind of protection mechanism built into the library.

8.1.4 Upgrades

When new releases of NPL become available, no special procedure is required to update the DLL's to run with the new version. However, Niakwa reserves the right to extend the functionality of the DLL specification on future releases, and older DLL's may require updating to take advantage of any such extended functionality.

8.2 Contents of the MS-Windows BESDK

The BESDK packages provided with the Niakwa Development Package is contained on a single diskette. The BESDK files for MS-Windows are stored on the MS-Windows Supplementary Files Diskette, and must be installed on a hard disk before they can be used. The INSTALLN batch file copies the contents of the diskette to a specified target directory (and subdirectories). Within the BESDK package, files 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 this text.

NOTE: The examples are provided to test versions of compilers, assemblers, linkers, etc.; being used with pretested source files and to help clarify any points that may be unclear in the text. It is recommended that customized versions of the BESDK examples be produced before starting a customized project, to ensure the various utilities work together as they should.

The following describes the contents of the MS-Windows BESDK.

\(root directory)	This directory contains all subdirectories pertaining to the operation of BESDK. It also contains the following files:
\README.DOC	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.
\INSTALLN.BAT	A batch program to install the BESDK files to a specified directory and subdirectories.

The remaining files are separated into the following subdirectories respectively:

\INCLUDE	This directory contains files that are common to all implementations or to all implementations of a specific language. It is recommended that the files in this directory not be changed.
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 from the MYMODULE.BS2 diskimage.
MYSTART.SRC	Source version of the NPL program used to test the example sub-routines and FUNCTIONs.
MYMODULE.SRC	Source version of the NPL library module used to specify the interface to the example sub-routines 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	An NMAKE script to compile MYBOOT.SRC into MYBOOT.OBJ. Assumes that MYBOOT.SRC and makefile are in the current directory and b2c can be accessed (the environment PATH is set to allow it to be found). To use, enter: "nmake".
RTPALL.H	Include file for Microsoft C programs, with structure and type definitions.
RTPALL.INC	For programmers that have BESDK libraries in Microsoft Assembler. Release IV features are not supported under this language.
RTPALL.PPI	For programmers that have BESDK libraries in Metaware Pascal. Release IV features are not supported under this language.
RTPALL.PI	For programmers that have BESDK libraries in Microsoft Pascal. Release IV features are not supported under this language.
\INCLUDE\WIN	Contains files that are specific to the MS-Windows environment. It is recommended that the files in this directory not be changed. The files provided in this directory are:
RTPDEFFN.H	Operating system-dependent macros to define the Microsoft C language attributes of external routines.
LIBMAIN.C	Start-up code required for all MS-Windows DLL's, written in C.
LIBMAIN.ASM	Start-up code required for all MS-Windows DLL's, written in MASM.
RTPPARM.OBJ	Compile version of RTPPARM.C using Microsoft C.

RTPPARM.C	C subroutines to provide the rtpfn_getparminfo() function used to check function declarations.
MAKEFILE	A script for the NMAKE utility to produce both mainline and customized DLLs. To run type: NMAKE
\WINCDIAL	Contains the files for the example that shows the use of call-back functions to implement Dialog boxes under the MS-Windows version.
WINCDIAL.TXT	Summary information about the example.
MYPROC.C	External FUNCTIONs and PROCEDUREs
MYRTPEXT.C	RTPEXT function directory code
MAKEFILE	NMAKE project make script
TESTDIAL.SRC	Symbols from TESTDIAL.DLG
TESTDIAL.DLG	Dialog generated by Resource Workshop
MYSTART.SRC	Sample program source
WINMISC.SRC	assorted symbols from WINDOWS.H
MYMODULE.SRC	Library interface module
MYPROC.H	FUNCTION and PROCEDURE interface info
MYICON_1.ICO	Binary icon used by test dialog
MYDIALOG.RC	Resource script for MYDIALOG.DLL
WINMSG.SRC	WM_xx symbols from windows.h
NPLDEFS.EXE	Convert .H to .SRC utility program
NPLDEFS.C	Convert .H to .SRC utility C source

MYDIALOG.DEF	Linker definition for MYLIB.DLL
MYLIB.DEF	Linker definition for MYLIB.DLL
MYDIALOG.DLL	Generated by the example - Contains dialog templates.
MYDIALOG.RWS	Generated by the example - Resource workshop project file.
MYLIB.DLL	External DLL needed to access Windows API subset.
MYBOOT.OBJ	NPL BOOT Program for WINCDIAL example
MYMODULE.BS2	NPL Diskimage for WINCDIAL example
\WINMEXAM	Contains example files for the MS-Windows implementations of external subroutines using Microsoft MASAM .
\WINCEXAM	Contains example files for MS-Windows implementations of external subroutines using Microsoft C.

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 Microsoft C programs
ASM for Microsoft MASM programs

MAKEMAIN.BAT	Batch file to produce the mainline for the example.
MAKEDLL.BAT	Batch file to produce the customized DLL for the example.

MYMAIN.DEF	A linker definition file that contains segment and operating system environment information used to make the mainline for the example.
MYLIB.DEF	A linker definition file that contains segment and operating system environment information used to make the DLL for the example.
MAKEFILE	Script for NMAKE utility to produce both mainline and customized DLL (all programs that are out of date are remade). To run it, type "nmake".

The \WINCEXAM directory also contains the following files, specifically for Release IV call back 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	Interface file containing parameter block specifications required by mycallbk.c
MYPROC.C	Source code to illustrate the implementation of the example external PROCEDURE in C.
MYPROC.H	Interface file containing parameter block specification required by myproc.c

NOTE: If the makefile is changed, delete all previously made .obj files in the directory before running NMAKE again.

The batch files and "makefile" NMAKE scripts assume:

- The compiler executables (such as CL, MASM, LINK, 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" (and ..\INCLUDE\WIN).

- All required system libraries are in the default directory specified by the LIB environment variable. All required system include files are in the default directory specified by the INCLUDE environment variable (required for both Microsoft C and Microsoft MASM).

The NMAKE script files assume the current (4.00.00 or later) version of the NPL compiler "B2C" is on the execution PATH.

8.3 Installation of the MS-Windows BESDK

The MS-Windows Supplementary Files Diskette has been produced in MS-DOS format on two different media: 5-1/4" 1.2MB diskettes and 3-1/2" 720K diskettes.

To install the MS-Windows BESDK, insert the MS-Windows Supplementary Files Diskette in the floppy drive, select the drive and directory to install the BESDK on and enter the following command for the appropriate drive being used:

To install from drive A:

```
A:INSTALLN A: .
```

To install from drive B:

```
B:INSTALLN B: .
```

The BESDK directories and files are extracted to the currently selected directory.

NOTE: The example DLL files are also installed with the MS-Windows BESDK files. Refer to Appendix C of this addendum for documentation on these example files.

8.4 MS-Windows Support

The external call features of NPL are supported in the MS-Windows environment as described in the following sections.

8.4.1 Environments

NPL is designed to run only in the protected mode environments of MS-Windows 3.1 (i.e., standard and enhanced modes). Large model addressing is recommended for use with BESDK, since a few code and data pointers are "far". Optimization of code in the DLL's may be possible by knowledgeable programmers using other models; however, this topic is beyond the scope of this document.

Operating system functions are accessed from the MS-Windows API, which defines a variety of application and operating system functionality for the MS-Windows environment. Most functionality available from MS-DOS interrupt 21H which does not affect the display or keyboard is also available within the MS-Windows environment. However, routines that interact with the user (from the display, keyboard or mouse) need to be rewritten to accommodate new restrictions imposed by the multi-tasking and graphical environment capabilities of the MS-Windows environment. At the time of writing, the MS-Windows environment is not well suited for development of actual MS-Windows applications--the DLL's must be compiled and linked under MS-DOS or an MS-DOS session under MS-Windows (memory permitting).

NOTE: Although several MS-Windows products do a reasonable job of making DLLs, BESDK make files and batch files will only run under MS-DOS, and the current recommended compiler (Microsoft C 7.0) ONLY runs in a DOS box.

8.4.2 Differences in the Flow Control Due to DLL Use

The use of DLL's for external functions has affected the flow control of operations (compared to that of MS-DOS and other environments). Under MS-Windows, NPL functions as the mainline task, and the external library is only called for specific localized functions. This means that some types of operations, (for example, subroutines that expect floating point exceptions, or inspect the environment list) which, in the MS-DOS environment, would normally rely on initialization performed by the C start-up routines, may require special handling to be usable in a DLL.

Because the externals no longer define the "mainline", initialization and cleanup functions which, under MS-DOS, are performed in the mainline (before and after calling the RTP() function), must instead be performed by the library start-up (LibMain) and shutdown (WEP) functions. When first loaded, the LibMain routine of the DLL is called. When the library is unloaded, the WEP routine of the DLL is called.

NOTE: If the libraries are configured for shared use, these procedures are only called once--when the first user loads the library (start-up) and when the last user unloads the library (WEP). Otherwise, libraries are not notified when a new user connects to the library (however, see notes on RTPEXT_SHAREABLE function, below).

8.4.3 Exported Symbols and Reserved Names

Use of DLL's requires familiarity with the concept of the "exported" symbols of a library. These are entry points to functions in the library that are available to users of the library. They must be explicitly listed in the linker definition (.DEF) file used to make the DLL.

For purposes of the MS-Windows BESDK, avoid the use of symbols that start with "RTPEXT", especially exported symbols and named resources, except where the use of such symbols is defined by the BESDK. The current implementation assumes that the existence of certain symbols of this type are for interface with NPL, and future revisions may extend the number of symbols of this type.

8.4.4 Debugging MS-Windows Applications

Debugging MS-Windows applications is best performed by the MS-Windows SDK debugger CVW (CodeView for MS-Windows). At the time of writing, CVW requires a system equipped with 2 monitors, typically a monochrome text monitor for the debugger display and a graphics display for the application.

8.4.5 Adapting MS-DOS Code for the MS-Windows Environment

Object code libraries targeted for the MS-DOS environment may sometimes be adapted for use under MS-Windows. However, applications that attempt to interact with the user from the keyboard, screen or mouse usually violate MS-Windows programming guidelines.

Code must also be capable of operating under the protected mode of a 286 processor (self-modifying code and execution of "built" code is generally not permitted).

Exported functions must begin with the MS-Windows prolog (a MOV AX,0nnnnH instruction in assembly language) to load properly as a DLL under MS-Windows (this instruction is patched at the time the DLL is loaded).

8.5 Loading the External Libraries

Under MS-Windows NPL applications can load external subroutine libraries (.DLL library created using BESDK) in two ways. In addition more than one library can be loaded for each application. The DLL's can be loaded as follows:

1. Specified as /X options on the command line.

   For example:

   ```
   RTIWIN /XLIBRARY1 /XLIBRARY2 MYBOOT
   ```

   loads both LIBRARY1.DLL and LIBRARY2.DLL., using the standard Windows DLL search procedure. Refer to the LoadLibrary Windows API call for details of this search order.

2. Specified as an ExternalLibraryx= FILENAME option in the RTIWIN.INI file in either the general or application's private section. Here, the value of x is a integer index number (without any leading 0's). Refer to Section 3.4 for more information.

If more than one ExternalLibraryx= option is entered in a section, the index (x value) must start at 1 and be consecutive.

NOTE: Where possible, the duplication of function names/numbers in external libraries should be avoided. However, this is not treated as an error.

When an external function or DEFFN' is declared in more than one external library, the order of loading becomes important. The specified libraries are loaded in the following order:

- Any libraries specified on the command line as an /X option
- Any libraries in the local RTIWIN.INI file's application section
- Any libraries in the NetworkIniFile's application section

- Any libraries in the local RTIWIN.INI file's [general] section
- Any libraries in the NetworkIniFile's [general] section

The first loaded library which contains a referenced function effectively hides any similarly named/numbered functions in subsequently loaded libraries. Named aliases for numbered DEFFN's are still visible if the numbered DEFFN becomes hidden.

When checking the DLL for resources or calling routines that permit modification of the NPL environment, libraries are checked "last-loaded first", so that the most local versions have the last chance at changes (and hence have priority). This applies to the following resources and exported functions:

- Any icon named RTPEXT_ICON,
- Any cursor named RTPEXT_CURSOR,
- Any accelerators named RTPEXT_ACCELERATORS,
- Calls to RTPEXT_LOGFONT;
- Calls to RTPEXT_MAIN;
- Calls to RTPEXT_SET_INPUT_SCREEN_ADDRESS;

For example, if the local RTIWIN.INI file contains:

```
[GENERAL]
NetworkIniFile= K:\PUBLIC\RTI.INI
ExternalLibrary1= C:\BASIC2C\NPLWIN.DLL
[xxx\MYBOOT.OBJ]
ExternalLibrary1= C:\BASIC2C\NPLDDE.DLL
ExternalLibrary2= C:\BASIC2C\NPLDLG.DLL
```

and the network .INI file (K:\PUBLIC\RTI.INI) contains:

```
[GENERAL]
External Library1= K:\NPLLIBS\NPLMSG.DLL
[xxx\MYBOOT.OBJ]
ExternalLibrary1= K:\NPLLIBS\NPLOLE.DLL
```

and the following command is run from the program manager:

```
RTIWIN /XLIBRARY1 /XLIBRARY2 MYBOOT
```

NPL loads the following libraries (in order of decreasing priority):

```
LIBRARY1.DLL
LIBRARY2.DLL
C:\BASIC2C\NPLDDE.DLL
C:\BASIC2C\NPLDLG.DLL
K:\NPLLIBS\NPLOLE.DLL
C:\BASIC2C\NPLWIN.DLL
K:\NPLLIBS\NPLMSG.DLL
```

8.6 Microsoft C under MS-WINDOWS

As shown in the example, writing external subroutines in Microsoft C for the MS-Windows environment is straight-forward. Examples assume Microsoft C version 6.0 or later. Earlier versions may also work, but are not tested.

The Microsoft C compiler adds an underscore ("_") to the start of all labels unless these are designated as using the "pascal" naming convention. In the following discussion, labels are presented the way they must appear in the source files of the C routines.

8.6.1 General

Use the large model "for MS-Windows" option (-ALw) on all "cl" compile commands. This ensures that all pointers are "far" unless specifically designated otherwise, and ensures that assumptions about the near data area and stack are appropriate for MS-Windows DLL's. In addition, the -Gw option is required to make sure the correct prolog for MS-Windows exported functions is used. Since the DLL is only used in protected mode, the -G2 option can also be used to allow the compiler to generate code that requires a 286 processor or better.

Make sure the include files provided with the BESDK are available in a directory specified by a "/Idirectory" option to the "cl" command that is defined as the standard .c to .obj inference rule.

8.6.2 Mainline

Unlike the MS-DOS quick-library implementation of external libraries, DLL files do not have a mainline. Instead, the code in the library is separated into 3 parts:

1. An initialization function (LibMain) that is executed once, when the library is loaded.

2. Various "exported" subroutines, including RTPEXT, which can be called once the library has been loaded.

3. An exit function (WEP) that is executed once, when the library is unloaded.

For purposes of testing subroutines without a MS-Windows RunTime, a user mainline for MS-Windows application must be written. The starting label of user code is called "WinMain".

NOTE: Substantial start up code from the Microsoft C library is executed before reaching the "WinMain" routine).

The standalone mainline, after performing appropriate initialization, calls a user-written RTP subroutine. The RTP subroutine should be referenced as an external with the standard BESDK calling conventions.

The RTP subroutine calls the external library functions (linked separately in the user's DLL) with appropriate test values.

8.6.3 Calling Conventions for BESDK Subroutines

Test RTP Subroutines

Declare all GOSUB' routines using standard BESDK calling conventions (i.e., the subroutine preserves non-volatile registers, arguments are pushed in the order they appear, arguments are popped by the called routine). The "rtpdeffn.h" include file for MS-Windows defines "rtpdeffn_ext" as equivalent to this designation.

RTPEXT Subroutine

The RTPEXT subroutine should be defined as a procedure with the standard BESDK calling 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). The name of this procedure must be exported, by including the name as an entry in the EXPORT section of the .DEF file used for the DLL linkage.

GOSUB' Subroutines

Use the standard BESDK calling conventions on declarations of all subroutines that are called from the GOSUB' interface. The "rtpdeffn.h" include file for MS-Windows defines "rtpdeffn_ext" as equivalent to this designation.

The name of all such procedures should be exported by including the name as an entry in the EXPORT section of the .DEF file used for the DLL linkage.

Formats of strings in NPL do NOT have a zero-terminator and are not of variable length. If strings are to be used by C library routines, make copies that have trailing spaces removed and a zero terminator added.

8.6.4 Linkage of Test Program

The files required for production of the standalone should include:

- The Mainline
- The RTP test subroutine
- The Microsoft C MS-Windows support libraries libw.lib, llibcew.lib (names may vary).
- An import library for the customized DLL.

To run, the standalone requires that the customized DLL containing the external functions is linked and available to the MS-Windows loader, either in the same directory as the standalone, or in a directory specified by the user's PATH specification.

8.6.5 Linkage of Customized DLL

The files required for production of the customized DLL should include:

- The LIBENTRY.OBJ module (Windows DLL library entry support file)
- The LIBMAIN.OBJ module (library start up and exit procedures)
- The RTPEXT subroutine (optional, but recommended)
- The GOSUB' subroutines
- Any function or procedure subroutines. If these are used the RTPPARM module (located in \INCLUDE\WIN) must also be used.
- The Microsoft C MS-Windows DLL support libraries libw.lib, ldllcew.lib (names may vary)

NOTE: Refer to Section 8.12 for an example of the use of callbacks to NPL functions.

8.7 Microsoft MASM Macro Assembler

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 libraries is usually small, 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. In addition, the programming interface to the MS-Windows API is written with the C programmer in mind, so familiarity with the requirements of mixed language calling conventions is a prerequisite for programming in assembler under MS-Windows.

Examples assume Microsoft MASM 5.10 or later. Earlier versions may also work but are not tested.

NOTE: External libraries written on Macro Assembler do not support the FUNCTION, PROCEDURE interface or callbacks to NPL.

8.7.1 General

Make sure the include files provided with the BESDK are available in a directory specified by a "/Idirectory" option to the "MASM" command.

8.7.2 Mainline

It is possible to write an entire standalone module in macro assembler.

- The RTP() subroutine should be referenced as a far external with the name "RTP".

- Use standard large-model conventions for segment names. Explicit segment names or the simplified segment directives supported by MASM 5.1 may also be used. Be sure that:

    1. All code segments have combine class "CODE".

    2. 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. Do not assume that DS and SS are in the same segment (they are not, in a DLL). Code should be written to conform to protected-mode requirements (never modify code segments or attempt to execute data segments, unless using segment aliases provided by the MS-Windows API for this purpose). The OFFSET operator applied to near-data items should always be based using DGROUP to ensure a correct address.

Where possible, use the MS-Windows API to perform functions of the operating system rather than attempting to control hardware directly. Procedures such as hooking interrupts are generally not permitted in the MS-Windows environment.

Exported functions must begin with the MS-Windows prolog (an MOV AX,0nnnnH instruction in assembly language) to load properly as a DLL under MS-Windows (this instruction is patched at the time the DLL is loaded).

8.7.3 Calling Conventions for BESDK Subroutines

Test RTP Subroutines

Declare RTP() subroutine as public with name "RTP".

Call GOSUB' subroutines using standard BESDK calling conventions (push arguments in order used in GOSUB' statements, assume arguments popped by subroutine, preserve used non-volatile registers).

RTPEXT Subroutine

The RTPEXT subroutine should be defined as using the default BESDK calling conventions 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 32-bit far (large model) 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 using the default BESDK calling conventions. When called, the parameters are on the stack below the return address. The first parameter of the GOSUB' is pushed first; last parameter pushed last.

A string parameter is passed as:

```
PUSH SEG < string>          ;16-bit segment
PUSH OFFSET < string>       ;16-bit offset
PUSH SIZE < string>         ;16-bit integer
```

NOTE: The string size is an unsigned 16-bit integer.

A numeric parameter is passed as:

```
PUSH SEG < rtpnum structure>          ;16-bit segment
PUSH OFFSET < rtpnum structure>       ;16-bit offset
```

The rtpnum structure is defined in the include file rtpall.inc.

To conform to standard BESDK 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).

8.7.4 Linkage of Test Program

Programs written in Macro Assembler must be linked to produce an executable file. All input files to "LINK" must be the result of previously run assemblies or libraries of files.

The files required for production of the standalone should include:

- The mainline (i.e., MYMAIN.OBJ)
- The RTP() test subroutine (i.e., MYRTP.OBJ)
- The Microsoft C MS-Windows support libraries LIBW.LIB, LLIBCEW.LIB (names may vary).
- An import library for the customized DLL.

8.7.5 Linkage of Customized DLL

The files required for production of the customized DLL should include:

- The LIBENTRY.OBJ module (Windows DLL library entry support file)
- The LIBMAIN.OBJ module (library startup and exit procedures)
- The mainline (i.e., MYMAIN.OBJ)
- The RTPEXT subroutine (i.e., MYRTPEXT.OBJ)
- The GOSUB' subroutines (i.e., MYSUB.OBJ)
- The Microsoft C MS-Windows DLL support libraries LIBW.LIB, LDLLCEW.LIB (names may vary)

8.8 Shared Data Segments in DLL'S

At the time of writing, the requirement for all MS-Windows DLL's is that the data segment of the DLL be a single SHARED data segment. This means that a DLL designed to be used by multiple tasks must be written carefully (even painstakingly) to ensure that no data specific to a caller is stored in the local data segment (unless this data has been specifically allocated on a per-task basis, controlled by handles passed by the caller, and so on). In particular, in C this means that care must be taken not to store information in global or static variables whose value is subsequently retrieved if, in the interim, another task could run (and overwrite the values stored).

To relax this requirement, NPL assumes by default that all DLL's are non-shareable, and makes physically distinct copies of the DLL under different names which then run with a single unshared data segment.

While this procedure avoids potential problems with code that is not designed to run with shared data, it does require more memory, since the code of DLL's used in this way is not shared. If the DLL contains a considerable amount of code, and can be written so that the shareable nature of the data does not pose a problem, this can be indicated to NPL by including an exported function RTPEXT_SHAREABLE in the DLL. The function has the following interface:

```
BOOL FAR PASCAL RTPEXT_SHAREABLE(int change_in_users);
```

where:

change_in_users is either 1 if the library is being accessed by a new NPL task, or -1 if the library is being released by an NPL task.

The routine should keep track of the total number of users accessed by NPL and return TRUE if the total number of users is within the sharing capability of the DLL.

For example:

```
#define MAX_USERS 4
static int total_users=0;

BOOL FAR PASCAL RTPEXT_SHAREABLE(int change_in_users)
{

    total_users += change_in_users;
    if(total_users <= MAX_USERS) return(TRUE);
    return(FALSE);
};
```

NOTE: The total number of users should be maintained whether the routine returns TRUE or FALSE. If the FALSE value is returned, NPL calls the routine again with a value of -1 to "uncount" the user.

To debug a .DLL, always define the RTPEXT_SHAREABLE function. Use a MAX_USERS value of 1, if the library cannot be shared. If this function is not defined, NPL uses a copy of the DLL with a different name, and any breakpoints you attempt to be set in the .DLL will never be executed.

8.9 Custom Resources in a DLL

When NPL loads a DLL for use by an application, it looks for resources with reserved names in the DLL, which is used to replace the standard resources normally used by RTI/RTP. These reserved names are:

An Icon named "RTPEXT_ICON"

If found, this icon is used for the application when minimized. If an icon is specified directly for the application in the user's RTIWIN.INI file, it takes precedence over this icon.

A Cursor named "RTPEXT_CURSOR"

If found, this cursor is used for the application.

An Accelerator table named "RTPEXT_ACCELERATORS"

If found, this accelerator table is used by the application.

NOTE: This capability would normally be used only if the main window menu is being replaced. This requires replacing or extending the main window's menu and subclassing the main application window.

For example, if the following MYLIB.RC script is defined:

```
RTPEXT_ICON ICON "MYICON.ICO"
RTPEXT_CURSOR CURSOR "MYCURSOR.CUR"
```

Then the command:

```
rc mylib.rc mylib.dll
```

instructs the resource compiler to insert the named resource files (created using the MS-Windows SDKPaint tool) into the DLL file.

8.10 Subclassing the Main NPL Window in a DLL

For heavily customized applications, it may be desirable to replace components of the standard "main window" used to present the NPL application. For such applications, NPL provides a hook function to inform the DLL of the handles used for the Instance and main display MS-Windows. If the DLL needs this information for any reason, this can be indicated to NPL by including an exported function RTPEXT_MAIN in the DLL. The function has the following interface:

```
void FAR PASCAL RTPEXT_MAIN(
                            HANDLE hLibraryInstance,
                            HANDLE hInstance,
                            HWND hMainWnd)
```

where:

hLibraryInstance is the module instance of the loaded DLL
hInstance is the Instance handle of the current task, and
hMainWnd is the window handle of the main NPL display window.

One use for the main window handle is as a parent window handle for pop-up windows that the DLL may have a need to display (containing error or other information). By making such windows "children" of the main display window, when the operator minimizes the main window, the children also become hidden, keeping the desktop relatively uncluttered.

For example:

```
static HANDLE hMainWindow;

BOOL FAR PASCAL RTPEXT_MAIN(
    HANDLE hLibraryInstance,
    HANDLE hInstance,
    HWND hMainWnd)
{
    hMainWindow=hMainWnd;                          /* save for later */
/* note : this use of static variable means the library is NOT share-
able! */
};
...
/* elsewhere in external subroutines where a parent is required */
    MessageBox(hMainWindow,                        /* child of main window */
                   "Divide by zero?",
                   "Error",
                   MB_OK);
```

Once these values are known, it is possible for the DLL to "subclass" the main window procedure, i.e., to intercept all window messages sent to the main window and interpret them in a different way. This technique might be used to replace the menu or add additional functions to the standard menu bar. Messages which are not intercepted should be passed on to the original window procedure.

Attempts to make this kind of customization can become very complex, and are discouraged unless the programming requirements of the MS-Windows environment are very familiar. Due to the nature of this kind of customization, it is not possible to guarantee upward compatibility of changes with future versions of NPL.

8.11 Flow Control for External Subroutines

The flow control (in chronological order) for the MS-Windows RunTime using external subroutines under MS-Windows is as follows:

1. NPL (from a LoadLibrary API call) informs MS-Windows that functions from the user's DLL is required. If the DLL is not already in use, it is located and loaded by MS-Windows and the LibMain function of the DLL is called. The LibMain routine may perform some initialization, and eventually returns (to MS-Windows).

2. NPL asks the DLL to determine if it is shareable. If an RTPEXT_SHAREABLE routine is exported from the library, it is called with a + 1 parameter to indicate a new task is using the library. If the routine does not exist, or returns a FALSE indication, indicating that it cannot be shared by another task, a copy is made, and the copy is loaded.

3. NPL runs and does some initial configuration work, including processing command-line options and loading the bootstrap program. Several MS-Windows customization entry points (RTPEXT_MAIN etc) are called if these are defined by the DLL. Custom resources (RTPEXT_ICON, RTPEXT_CURSOR) are selected if defined by the DLL.

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 numbered externals is built.

5. The NPL execution proceeds. At some point, a GOSUB', (e.g., GOSUB'100) is executed, and no local GOSUB' subroutine is found. If the GOSUB' to a named DEFFN' and the identifier is found in the table 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 (e.g., 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, an 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) does its thing 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.

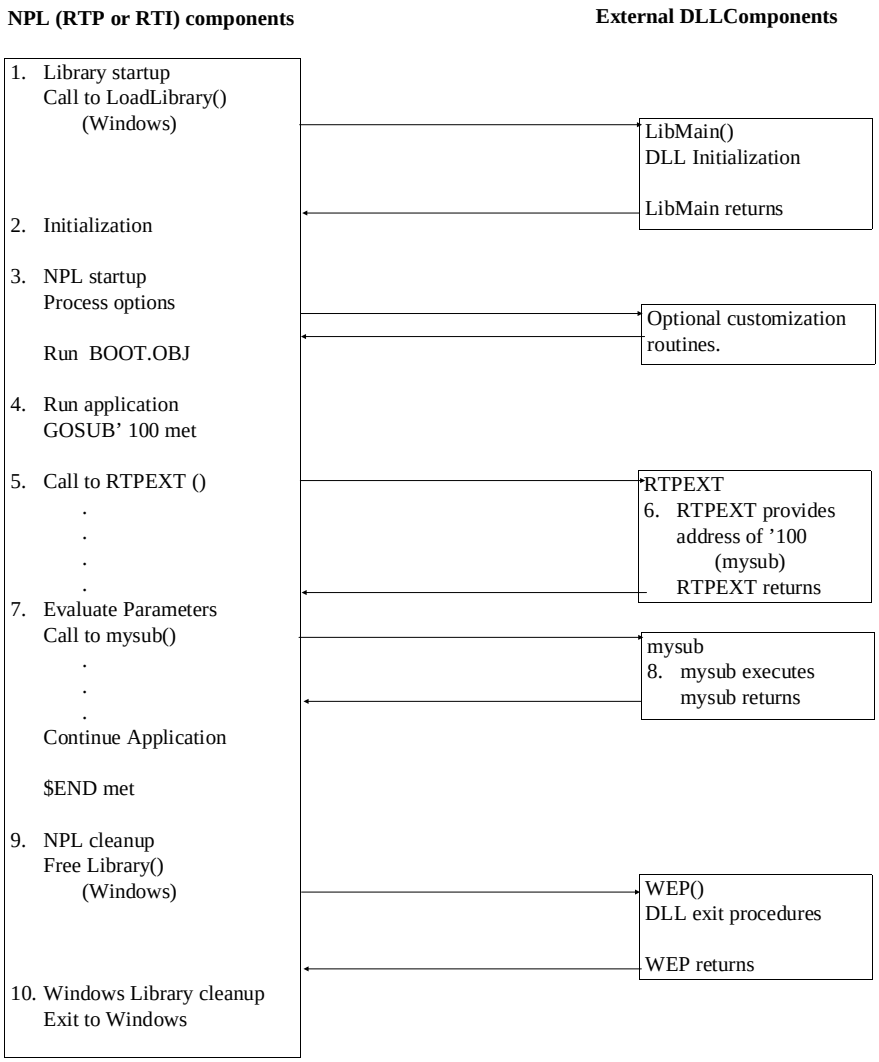
10. NPL does its cleanup. It calls RTPEXT_SHAREABLE (if defined) with a -1 parameter, to indicate a task is giving up use of the library, and notifies MS-Windows (from the FreeLibrary API call), that the DLL is no longer required. If no other MS-Windows tasks have registered an interest in the DLL, MS-Windows calls the DLL's WEP (Windows Exit Procedure) routine before removing the DLL code from the environment.

11. NPL does MS-Windows C library shutdown, and eventually exits back to MS-Windows.

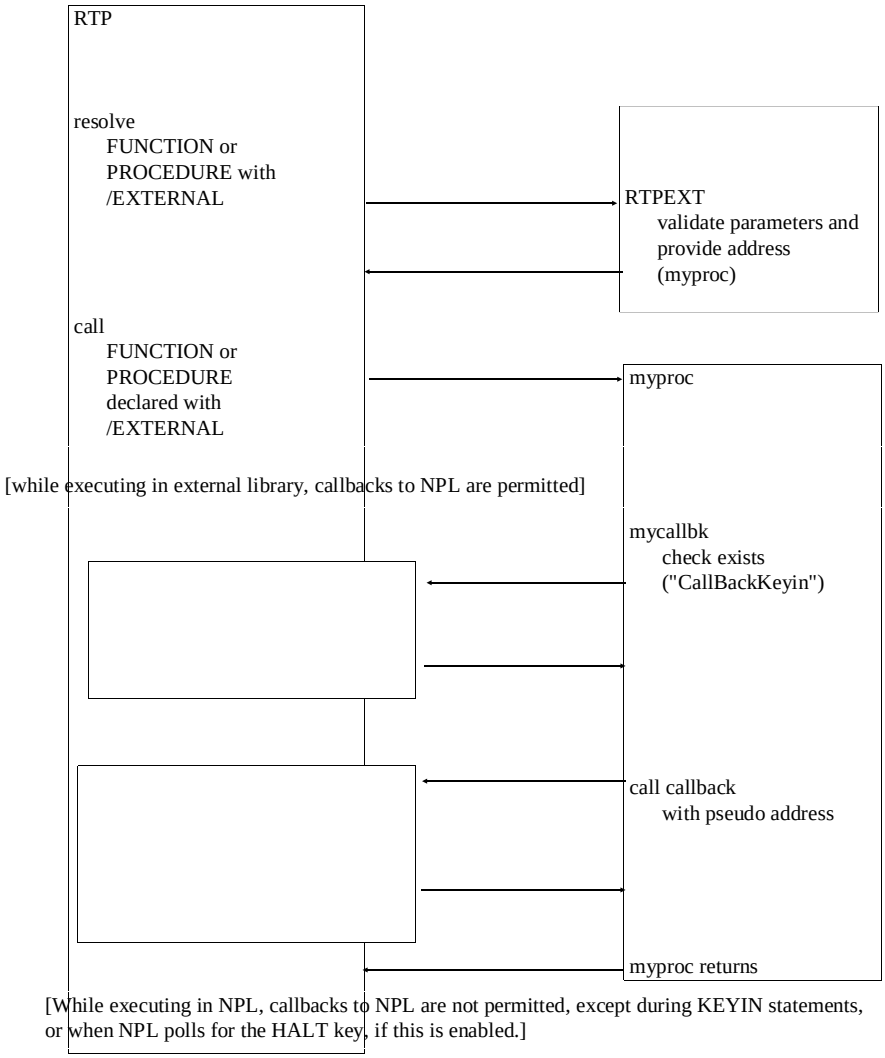
NOTE: RTPEXT can be called by LIST' to find out information about DEFFN' subroutines without calling the subroutines.

Subroutines should not DEPEND on the above flow control order to work (e.g., a subroutine should not expect RTPEXT to always be called immediately before it). The above outline is provided merely as a guide to the understanding of how the external mainline, rtp, RTPEXT and external subroutines interact.

The following diagram shows how execution proceeds using the various software components with the above steps labeled.


NPL (RTP or RTI) components
External DLLComponents
1. Library startup
Call to LoadLibrary()
(Windows)
LibMain()
DLL Initialization
LibMain returns
2. Initialization
3. NPL startup
Process options
Optional customization routines.
Run BOOT.OBJ
4. Run application
GOSUB' 100 met
5. Call to RTPEXT ()
RTPEXT
6. RTPEXT provides address of '100 (mysub)
RTPEXT returns
7. Evaluate Parameters
Call to mysub()
mysub
8. mysub executes
mysub returns
Continue Application
$END met
9. NPL cleanup
Free Library()
(Windows)
WEP()
DLL exit procedures
WEP returns
10. Windows Library cleanup
Exit to Windows

The following diagram shows the execution flow for the FUNCTION/PROCEDURE (interface):



8.12 Callbacks to NPL under MS-Windows

The MS-Windows RunTime permits callbacks to NPL from externals in two additional key locations:

1. When executing a KEYIN statement (either polling or wait version).

2. When the RunTime internally polls for a HALT (CTRL-BREAK) key.

No programming changes are required for this. This change allows callbacks to the Run-Time to occur on an "interrupt" like basis, as a result of messages dispatched to windows which may be created by the external routine.

NOTE: When NPL is in immediate mode, in HELP, or processing a LINPUT command, callbacks are disabled.

The MS-Windows RunTime does not poll the message queue to check for a HALT (CTRL-BREAK) key when the main window has been disabled (which occurs if a modal dialog box is created as a child of the main window). Also, when a task-modal or system-modal dialog box is displayed, the RunTime only polls the message queue to check for a HALT (CTRL-BREAK) key messages directed to the main window or its children.

This allows modal dialog box functions to call back to NPL, without being concerned that the polling mechanism will incorrectly get messages from the queue that would normally be picked up by the MS-Windows internal modal dialog box mechanism.

If a modal dialog box function calls back to NPL, the HALT key is not operational for the duration of execution of the called function, and as such cannot be used to terminate software bugs such as infinite loops.

Some operations (such as KEYIN, INPUT, LINPUT, STOP, END statements, and screen output with pauses due to SELECT P) require NPL to get messages from the queue.

WARNING--These can hang the RunTime if issued during execution of NPL code during a callback from a child dialog box.

Executing these operations while in a callback from a non-child dialog box function, may cause loss of messages (and incorrect operation) of controls in the dialog, and so should be avoided.

APPENDIX A

COMMON PROBLEMS

A.1 Overview

This Appendix details general problems that may be encountered when attempting to execute the NPL MS-Windows RunTime Program. Refer to Appendix A of the MS-DOS Supplement for further details on common problems encountered with the standard Niakwa RunTime.

A.2 Problems

Problem 1:
RTIWIN or RTPWIN fails the security check.

Possible causes: NIAKSECx not properly loaded.

Solution: Load the appropriate NIAKSECx file. If this is any file other than NIAKSECA.COM, an entry must be made in the RTIWIN.INI to ensure proper loading of this file. The NIAKSECx file must be loaded prior to starting MS-Windows--it cannot be loaded using the MS-DOS shell from within MS-Windows.

Problem 2:

The font size is not changing when window sizes are changed.

Possible causes: The Autosize option is not turned on or a font that does not support re-sizing is in use.

Solution: Check the option selection to verify that Autosize is selected. If it is, make sure the BASFONTS.FON file, IBMFONT.FON file, or another resizable fixed pitch font file is available and selected from the MS-Windows Control Panel font selection options.

Problem 3:

The message "BASFONTS.FON not found" is displayed.

Possible causes: The Niakwa font file, BASFONTS.FON, was not found by MS-Windows.

Solution: Make sure the BASFONTS.FON file is available and selected from the MS-Windows Control Panel font selection options.

Problem 4:

$OPEN does not work.

Possible causes: The MS-DOS SHARE command is not loaded and the Share Warning message has been ignored or suppressed by the use of **Share-Warning= 1** parameter in the RTIWIN.INI file.

Solution: Load the MS-DOS SHARE program as documented in Section 3.2

Problem 5:

The MS-Windows RunTime task does not execute correctly after initially loading the MS-Windows RunTime.

Possible causes: The security TSR is other than NIAKSECA, which the RunTime assumes is the default when the MS-Windows RunTime is loaded.

Solution: The RTIWIN.INI file must be created manually in the directory where MS-Windows is located by using a standard text editor (i.e., MS-Windows Notepad, EDIT, EDLIN, etc.). Create the general section of this file as documented in the example in Section 3.4 of this Addendum with the NIAKSECx set to the correct value, B-F for the interrupt on the host system.

Problem 6:

The error message, "Partition X mapped to Network partition y already used, cannot start this application" occurs when starting a RunTime task.

Possible causes: An attempt was made to execute a task that did not have a reserved Partition value assigned to it.

Solution: Any RunTime task that will be executed on a particular workstation, must be set up in the users local RTIWIN.INI file with a Partition option value. Refer to Section 6.2.2 for more information.

APPENDIX B

MODIFYING NPL FONTS

B.1 Overview

This appendix describes the processes used to modify or create new fonts for use with the MS-Windows RunTime.

The font modification procedure requires the tools provided with the MS-Windows SDK. The make script "makefile" requires that the utilities RC.EXE and LINK (not version 5.10) be accessible in the standard PATH.

NOTE: Version 5.10 of Microsoft LINK has a bug that prevents it from properly building the font file. Either newer or older versions of LINK may be used.

B.2 Installation

The files necessary for modifying or creating new fonts for use by the MS-Windows Run-Time are found on the MS-Windows Supplementary Files Diskette in the \BFONTS and \IFONTS directories. The contents of the appropriate directory should be copied to the directory where the font files are to be modified.

NOTE: Two sets of fonts files are included with the MS-Windows RunTime. They are the BASFONTS.FON font file and the IBMFONTS.FON font file. Refer to Chapter 1 for more details on these files.

B.3 Files

The following files are provided with the MS-Windows Supplementary Files Diskette in the \BFONTS directory for the BASFONTS.FON font file and \IFONTS for the IBMFONTS.FONT font file:

xxxFONTS.DEF	Linker definition file, required to produce a resource executable.
xxxFONTS.RC	The resource script specifying the names of fonts included in the font resource file.
FONTS.OBJ	A null object, required by LINK to produce a resource executable.
MAKEFILE	Instructions for nmake utility on how to produce the BASFONTS.FON and IBMFONTS.FON files.
B2fwXh.FNT	The individual font files that make up the BASFONTS.FON or IBMFONTS.FON file. These files are described by "w" wide and "h" high (in pixels)

where:

xxx = BAS for NPL (BASIC-2C) Fonts and IBM for the IBM fonts.

f = C for the BASFONTS.FON fonts and I for the IBMFONTS.FON fonts

B.4 Modifying Existing Fonts

The MS-Windows SDK FONTEDIT.EXE utility may be used to modify any of the included B2fwXh.FNT files.

To make the fonts usable with MS-Windows, run "nmake" in the current directory. The result is a new BASFONTS.FON or IBMFONTS.FON file. This file replaces the version shipped with the MS-Windows RunTime.

WARNING--Always keep the original BASFONTS.FON and IBMFONTS.FON file in a safe place as a backup.

B.5 Creating New Fonts

The MS-Windows SDK FONTEDIT.EXE utility may be used to create new fonts. The easiest way to do this is by modifying one of the existing B2CwXh.FNT or B2IwXh.FNT files.

For example, to add the font file to BASFONTS.FON, follow the steps shown below:

1. Make sure the individual font files are saved using the "Save As" option specifying that the font file is to be saved as Windows 2.0 compatible file.

2. Include the new file name in the list of .FNT files specified in the makefile dependency list.

3. Add the new .FNT file name to the BASFONTS.RC or IBMFONTS.RC file on a new line. The number at the start of the line should be the next in sequence (i.e., 17).

APPENDIX C

EXAMPLE DYNAMIC LINK LIBRARIES

C.1 Overview

To help NPL developers access the MS-Windows resources, a series of examples have been provided. These examples include an MS-Windows Clipboard example, text display routine, a message box routine, and a dialog box routine.

NOTE: These examples are included for purposes of example only and are not intended to be suitable for use by end users without additional work. End-user use of these routines is not supported by Niakwa.

Section C.2 discusses installation of the examples.

Section C.3 discusses the WINCDEMO examples.

Section C.4 discusses the WINCDIAL examples.

Development of external routines to access MS-Windows resources requires considerable expertise in use of C, ASM, and the MS-Windows SDK.

C.2 Installation

The example DLL files are found in the \WINCDEMO and \WINCDIAL directory of the MS-Windows Supplementary Files Diskette. These files are automatically installed when the MS-Windows BESDK is installed. Refer to Chapter 8 of this Addendum for information on installing the MS-Windows BESDK.

C.3 The WINCDEMO Example

The following example discusses the example DLL files and their implementation included on the MS-Windows Supplementary Files diskette.

C.3.1 Implementation Notes

The examples provided with the MS-Windows RunTime were implemented from external calls to "C" or ASM. Executable versions of all example external subroutines are included in the library MYLIB.DLL. Also included are example NPL programs that demonstrate the use of the example routines and the necessary parameters that must be passed to them for proper execution.

The following example Niakwa object programs are provided:

CLIP.OBJ	Demonstrates text interchange with the MS-Windows Clipboard. Text can be moved both ways from NPL to MS-Windows and the reverse, using MS-Windows functions like Copy and Paste.
TEXT.OBJ	Example of Text Box feature of NPL for MS-Windows. This program displays text in very large fonts (non-Niakwa) inside Window type of boxes.

MESSAGE.OBJ	This demonstrates the use of text boxes and buttons with NPL.
DIAGBOX.OBJ	Illustrates the use of MS-Windows dialog boxes within NPL, allowing the user to make a choice and bring the result directly into the NPL environment.

C.3.2 Example DLL Files

The following is a complete list of the example DLL files provided:

\WINCDEMO

MAKEFILE	Instruction for the nmake utility on how to produce the .DLL and stand alone MS-Windows executable.
DLGOPEN.ASM	Assembly language helper routines for DLGOPEN.C.
DLGOPEN.C	The routines to display a standard file/open and file/save dialog box.
LIBMAIN.C	Source code for entry and exit procedures required by all MS-Windows DLLs.
MYCUSTOM.C	Source code for MS-Windows customization routines, described in Chapter 8 of this Addendum.
MYMAIN.C	Source code for mainline of standalone MS-Windows executable used to test the .DLL without RTI. Calls RTP() function after normal MS-Windows start up.
MYRTP.C	Source code for a RTP() routine that calls .DLL with values to test the performance of the .DLL, for use with the standalone test program.
MYRTPEXT.C	Source code defining the available routines in the .DLL and the parameters required by each.
MYSUB.C	Source code for the majority of functions provided by the .DLL.
MYCURS.CUR	A custom cursor resource file, created and modifiable using SDKPaint.

MYMAIN.DEF	Linker definition file, required to produce the standalone MS-Windows executable MYMAIN.EXE.
MYLIB.DEF	Linker definition file, required to produce the dynamic link library MYLIB.DLL.
MYRES.DLG	Resource script for the File Open dialog box.
MYLIB.DLL	Pre-made copy of the .DLL to allow use by users who do not have the MS-Windows SDK.
DLGBOX.H	Identifying constants for the File Open dialog box control.
DLGOPEN.H	Constants and interface specification for the File Open dialog box routines.
MYICON.ICO	A custom icon resource file, created and modifiable using SDKPaint.
MYRES.RC	The resource script specifying the names of icons, cursors, etc., included in the .DLL resource file.

NOTE: The C programs are based on the MS-Windows SDK example programs.

C.3.3 Starting the Example Programs

To start the example programs, follow the steps shown below.

1. After installing the MS-Windows BESDK files (refer to Chapter 8 of this Addendum), start MS-Windows.

2. Add a new program item to the MS-Windows Program Manager (as described in Section 3.3 of this Addendum) and specify the Command Line entry as:

   ```
   RTIWIN.EXE /XC:\NPL4\MYLIB.DLL BOOT
   ```

   Substitute NPL4 with the proper directory location, if necessary. Specify the working directory as the WINCDEMO directory. BOOT is the name of the boot program to be used (i.e., CLIP, TEXT, MESSAGE, DIAGBOX), including the path if different from the directory WINCDEMO directory.

3. Start the task.

4. To run the example enter: Return. To view the Niakwa source code, invoke the Niakwa HELP processor and select the Reset option (or HALT the program by pressing CTRL-BREAK).

C.3.4 The DLL Examples

Before beginning work with any of the example DLL's, it is strongly recommended that the standard BESDK example be completed (refer to Chapter 8 of the Addendum for this example). This confirms that all components are working correctly before attempting to use the MS-Windows resources.

NOTE: All strings passed to the .DLL should have at least one trailing space--quotes included.

Clipboard Example

The use of the Clipboard feature is contained in the boot program CLIP.OBJ. This program contains two calls to the external subroutines: SetClipboardText and GetClipboardText. These are defined as follows:

NOTE: The literal subroutine names included in the example, are for documentation purposes only. The subroutine number (i.e, 10002) is all that is necessary to access the example routines.

```
GOSUB'10001 SetClipboardText (Text$, Result)
```

Where the parameters are defined as follows:

Text$ The text to be passed to the MS-Windows Clipboard.

Result Success/failure codes. There are many potential failures, but only a 0 means success.

Dialog Box Example

The dialog box example program is called DIAGBOX.OBJ and the parameters necessary for the external call used to create the dialog box are described below.

```
GOSUB'10004 DialogBoxFile(Title$, FileIn$, Ext$, FileOut$, Result)
```

Where the parameters are defined as follows:

Title$ The caption (name of the dialog box) displayed at the top of the dialog box.

FileIn$ The default file name to highlight (not implemented).

Ext$ The default file extension used as the search parameter for the filename.

FileOut$ The file selected.

Result Success/failure codes. There are many potential failures, but only a 0 means the box actually is displayed.

This example demonstrates how a file could be selected from a list provided inside a MS-Windows dialog box.

Message Example

This program demonstrates the use of the MS-Windows message box. The external subroutines calls from MESSAGE.OBJ are described below.

```
GOSUB'10000 MessageBox (Caption$, Text$, Buttons, Icon, Default, Result)
```

Where the parameters are defined as follows:

Caption$ The caption name to appear in the message box.

Text$ The text to be displayed--it is not necessary to specify where line breaks should go, the text appears without splitting words across lines, although multiple lines may be explicitly indicated by inserting CR/LF's.

Buttons The button or buttons to be displayed in the message box.

Possible values for the Buttons parameter are:

Button Displayed	Value
OK	0
OK/Cancel	1
Abort/Retry/Ignore	2
Yes/No/Cancel	3
Yes/No	4
Retry/Cancel	5

Icon The icon(s) to appear in the message box.

Possible values for the Icon parameter are:

Icon Displayed	Value
No icon	0
Hand	1
Question Mark	2
Exclamation Point	3
Asterisk	4

Default The default button.

Possible values for the Default parameter are:

Default	Value
Button 1	0
Button 2	1
Button 3	2

Result Return code - indicates which button was selected.

For example, to get a box that has yes/no buttons, a hand (STOP sign icon and default to the NO button, set the parameters as follows:

Buttons 4
Icon 1
Default 1

```
GOSUB'10000 ("Example message box ", "This is a message box with
Yes/No buttons, a stop sign, with the No button defaulted ", 4,1,1,4)
```

for the type field in the GOSUB' call.

The return value specifies which button was selected as shown below:

Button Selected	Return Code Value
OK	1
Cancel	2
Abort	3
Retry	4
Ignore	5
Yes	6
No	7

NOTE: Additional parameters may be used to allow access to other Message Box features such as the ability to switch to other windows while the message box is displayed.

Text Box Example

The text box example is contained in the program TEXT.OBJ. This contains one call to the external subroutine which displays the text using fonts other than the normal MS-Windows RunTime screen display font. This call, shown below, calls the external routine that produces a box with text inside.

```
GOSUB'10005 MakeTextBox( Title$, Text$, LogFont$, X++, Y++, Child, Bg,
Fg, HandleOut$, Result)
```

Where the parameters are defined as follows:

Title$ If not blank, this text appears in a box with a border and the standard windows caption line (and the box is moveable). If blank, no border, or caption is used and the box is not movable.

Text$ The text to be displayed - it is not necessary to specify where line breaks should go, the text appears without splitting words across lines, although multiple lines may be explicitly indicated by inserting CR/LF's.

LogFont$ This is a 50-byte string modeled on the Windows structure used to select a logical text font, as specified below (there is more detail provided in the MS-Windows SDK documentation). Each "int" field occupies 2 bytes, ordered high/low (same as BIN(,2)).

The FaceName must have a null byte terminating it.

```
LF_FACESIZE 32
struct {
STR(,01,2)    int       lfHeight;
STR(,03,2)    int       lfWidth;
STR(,05,2)    int       lfEscapement;
STR(,07,2)    int       lfOrientation;
STR(,09,2)    int       lfWeight;

/* Font Weights */
        FW_DONTCARE           0
        FW_THIN                             100
        FW_EXTRALIGHT         200
        FW_LIGHT                            300
        FW_NORMAL                           400
        FW_MEDIUM                           500
        FW_SEMIBOLD           600
        FW_BOLD                             700
        FW_EXTRABOLD          800
        FW_HEAVY                            900
STR(,11,1)    BYTE      lfItalic;
STR(,12,1)    BYTE      lfUnderline;

STR(,13,1)    BYTE      lfStrikeOut;
STR(,14,1)    BYTE      lfCharSet;
        ANSI_CHARSET          0
        SYMBOL_CHARSET        2
        SHIFTJIS_CHARSET      128
        OEM_CHARSET                         255

STR(,15,1)    BYTE      lfOutPrecision;
        OUT_DEFAULT_PRECIS                  0
        OUT_STRING_PRECIS                   1
        OUT_CHARACTER_PRECIS                2
        OUT_STROKE_PRECIS                   3

STR(,16,1)    BYTE      lfClipPrecision;
        CLIP_DEFAULT_PRECIS                 0
        CLIP_CHARACTER_PRECIS 1
        CLIP_STROKE_PRECIS                  2


STR(,17,1)    BYTE      lfQuality;
        DEFAULT_QUALITY                     0
        DRAFT_QUALITY                                 1
        PROOF_QUALITY                                 2

STR(,18,1)    BYTE      lfPitchAndFamily;
/* Pitch defined in low 4 bits of this field */
        DEFAULT_PITCH                                 0
        FIXED_PITCH                                   1
        VARIABLE_PITCH                                2
```

```
/* Font Families */
         FF_DONTCARE  (0<<4)/* Don't care or don't know. */
         FF_ROMAN     (1<<4)  /* Variable stroke width, serifed. */
                              /* Times Roman, Century Schoolbook, etc. */
         FF_SWISS      (2<<4)  /* Variable stroke width, sans-serifed. */
                                                    /* Helvetica, Swiss, etc. */
         FF_MODERN     (3<<4)  /* Constant stroke width, serifed or sans-serifed. */
                                                    /* Pica, Elite, Courier, etc. */
         FF_SCRIPT     (4<4)  /* Cursive, etc. */
         FF_DECORATIVE    (5<4)  /* Old English, etc. */

STR(,19,32)   BYTE      lfFaceName[LF_FACESIZE];
```

X — The horizontal coordinate of the left edge of the box. After the call, this variable receives the coordinate of the right edge of the box.

Y — The vertical coordinate of the top edge of the box. After the call this variable receives the coordinate of the bottom edge of the box.

Child — If this flag is 0, the box is a "pop-up" (may exceed the boundaries of the MS-Windows RunTime main window) and the X and Y coordinates are relative to the top left of the whole screen.

If this flag is non-0 the box is a "child" (stays inside the MS-Windows RunTime main window) and the X and Y coordinates are relative to the top left of the main window. Child boxes move when the MS-Windows RunTime main window is moved. This includes any scrolling operations so, in immediate mode, the boxes can "roll off" the screen.

Bg — Background color (MS-Windows RunTime color conventions).

Fg — Foreground color (MS-Windows RunTime color conventions) for text.

HandleOut$ — The magic window handle (2 bytes) returned by the function. This value must be known to remove the window.

Result — Success/failure codes. There are many potential failures, but only a 0 means the box actually is displayed.

The following routine removes the previously generated text box.

WARNING--No other NPL operation (CLEAR, etc.) will remove the Text Box. Do not lose the value of the magic handle. Even if the Text Box scrolls off the screen, it must be killed or MS-Windows can run out of memory (eventually).

```
GOSUB'10006  KillTextBox(Handle$,Result)
```

Handle$ The handle value returned from MakeTextBox must be supplied here.

Result Result 0 means success.

C.3.5 Programmer's Notes

Windows maps the logical font to the closest available installed font, consequently, the font is not always exactly what is expected. Windows may "blow up" a smaller font to the requested size, so asking for huge fonts may look nice, but may be totally inappropriate, depending on what fonts are available.

The treatment of X and Y parameters is intended to at least partly simplify the arrangement of text boxes when there are a number of them displayed at once.

There are some items missing from the example program codes like declaring the size of the box, but as noted before, these programs are not intended as working programs, just examples. Currently, the size of the box is the largest (horizontally and vertically) that fits on the screen given the initial X and Y coordinates and the framing environment (full screen or MS-Windows RunTime window), trimmed to the amount actually used by the text. The actual windows function used to display the text is the DrawText() function. This function also has a number of options (like centering lines) that might be of use but aren't accessible at this time.

In addition, options to move or modify the size of a window (but not delete it) once it is created (i.e., move it, change size, change text display options/ fonts, hide/ show it) could be added to these examples.

C.4 WINCDIAL Example

This example BESDK library shows non-trivial use of callback functions to implement Dialog boxes under the windows version.

The WINCDIAL example is automatically installed by the MS-Windows BESDK installation program INSTALLN. Refer to Section 8.3 for details. The files are placed in the WINCDIAL directory which is established at the same level as the WINCEXAM directory.

All code necessary to execute this example is included. This allows the execution of the example without "making" it. In addition, all source files used by the example are included.

To execute the example, start RTIWIN with the /X option specifying:

```
RTIWIN /X\NPL4\WINCDIAL  MYBOOT
```

Substitute NPL4 with the proper directory location if necessary. Specify the working directory as the WINCDIAL directory.

WARNING--Do not attempt to STEP through the call to 'DialogBoxParam at line 1800. This will cause the task to hang. If this occurs, it is necessary to exit Windows and begin again.

With this example, a sample dialog box (designed using Borland's Resource Workshop) is processed which contains a number of types of controls, whose values can be set and read using the support API. The dialog box is repeatedly shown until the CANCEL button is pressed/clicked.

The main function provided by the DLL is the function DialogBoxParam(), which allows access to any predefined dialog box which is currently loaded in a .DLL.

A number of other helper functions are implemented to allow access to relevant API entries under windows needed to load the dialog template, and access the dialog controls. NPL versions of some of the relevant constants defined in "windows.h" for C programs are defined in libraries also.

The normal procedure to access the dialog is illustrated in the mainline routine of MYSTART:

1. Define an NPL callback function to handle dialog box messages. The parameters must be the same as defined in the 'DialogFunc() function defined in the MYMODULE library. It must be a /PUBLIC function.

2. Load the .DLL with the 'LoadLibrary function. The .DLL must be in the current directory, or in one of the various places windows looks for .DLL's.

3. Execute the dialog box with the 'DialogBoxParam function, specifying the name of the NPL function defined in (1) to handle dialog messages.

4. The return code indicates how the dialog box was completed, in this case the value should be IDOK or IDCANCEL depending on which button (OK or CANCEL) caused the dialog to complete.

While the dialog box is popped up, many messages will be sent to the callback routine. The flow of messages here will be familiar to windows programmers.

All messages normally passed to a C dialog box function are passed back to NPL, except WM_SETCURSOR. This is screened out only because it is rarely handled differently from the default, and constitutes a large part of the volume of message traffic.

For example:

A WM_INITDIALOG message allows the dialog to set the initial values of controls.

A WM_COMMAND message is sent when buttons are clicked or various other edit controls are changed. The OK and CANCEL buttons cause the dialog to terminate by issuing 'EndDialog calls. Before terminating due to the OK button, the values of various controls are obtained and stored in module-visible variables.

A WM_HSCROLL message is sent when a change occurs to a horizontal scroll bar.

A WM_VSCROLL message is sent when a change occurs to a vertical scroll bar.

C.4.1 Notes:

As the need to customize the DLL's increases, access to more of the MS-Windows APIs will be required. For example, although you could define a dialog box with ListBox, ComboBox or UserDraw controls, properly supporting these controls in the dialog procedure would undoubtedly required using Window APIs which are not available with this example.

As indicated in the comment on line 1800 of the MYSTART program, if you intend to debug the dialog box procedure you MUST change the example so that the dialog box is NOT a child of the main window. This is accomplished by uncommenting (removing the semicolon from) the following line:

```
;hWndParent=NULL
```

This would normally be done only as a last resort in order to look at a specific message, if something is going wrong. In particular, STOP or other break-points should be placed so that only the specific messages of interest are intercepted.

Since making the dialog box not a child of the main window can introduce other problems related to message interception, the code should be changed back when the problem has been located and corrected.

The Borland workshop (and others) normally maintains an include file containing identifiers for the various controls. Included in the example is the source and executable of a small C program (npldefs) which converts this to an NPL source file, which is compiled into the test diskimage by the make script.

The MYDIALOG.DLL could in principle contain any number of application dialog box templates, and other applications could use other DLL's. Each dialog box template requires that an NPL dialog box function be written for it.

The same MYLIB.DLL could be used by all applications, and the same MYMODULE library module would be used by all applications that use the external.

Conventions for splitting the various NPL programs into application, library and system library diskimages have not been established - currently everything is just compiled into the MYMODULE.BS2 diskimage.

NOTE The current example dialog box is defined as a child of the RTI window. This means that you may not switch focus using the mouse to "get at" the NPL main window while the dialog is active. See the above instructions for debugging the dialog box procedure.

The 'lpParam$' parameter of DialogBoxParam would normally be used to pass initial values to the dialog box, and possibly to pass results back, but this is not implemented on the current cut.

If you are debugging the dialog box procedure, and if NPL is halted inside the NPL Dialog function, button clicks and keystrokes on the dialog box are handled using the default dialog box function - they are not saved in any queue for later processing by the NPL dialog function.

C.4.2 Description of Files in the Project

The following files are included with this example:

Source Files

WINCDIAL.TXT	Project summary information
MYPROC.C	External FUNCTIONS and PROCEDUREs
MYRTPEXT.C	RTPEXT function directory code
MAKEFILE	NMAKE project make script
TESTDIAL.SRC	Symbols from TESTDIAL.DLG
TESTDIAL.DLG	Dialog generated by Resource Workshop
MYSTART.SRC	Sample program source
WINMISC.SRC	Assorted symbols from WINDOWS.H
MYMODULE.SRC	Library interface module
MYPROC.H	FUNCTION and PROCEDURE interface info
MYICON_1.ICO	Binary icon used by test dialog

MYDIALOG.RC	Resource script for MYDIALOG.DLL
WINMSG.SRC	WM_xx symbols from windows.h
NPLDEFS.EXE	Convert .H to .SRC utility program
NPLDEFS.C	Convert .H to .SRC utility C source
MYDIALOG.DEF	Linker definition for MYDIALOG.DLL
MYLIB.DEF	Linker definition for MYLIB.DLL
MYBOOT.OBJ	NPL boot program
MYMODULE.BS2	NPL project diskimage, includes:

MYSTART	Sample application module
TESTDIAL	Control id numbers for TESTDIAL dialog
MYMODULE	INCLUDE module required to use MYLIB.DLL
WINMSG	INCLUDE module with windows message symbols
WINMISC	INCLUDE module with windows miscellaneous symbols

Externally Generated Files

MYDIALOG.RWS	Resource workshop project file

Intermediate Files Generated by Project Make

LIBMAIN.C	Same as \INCLUDE\WIN\LIBMAIN.C
RTPDEFFN.H	Same as \INCLUDE\WIN\RTPDEFFN.H
MYDIALOG.MAP	This is the load map generated by LINK when creating the MYDIALOG.DLL library.
MYLIB.MAP	This is the load map generated by LINK when creating the MYLIB.DLL library.
LIBMAIN.OBJ	This is the Microsoft compatible object code generated from compiling the MYLIB.C module.

MYPROC.OBJ	This is the Microsoft compatible object code generated from compiling the MYPROC.C module
MYRTPEXT.OBJ	This is the Microsoft compatible object code generated from compiling the MYRTPEXT.C module
RTPPARM.OBJ	This is the Microsoft compatible object code generated from compiling the RTPPARM.C module
MYDIALOG.RES	This is an intermediate compiled resource file generated by RC (the resource compiler)

Product Files Generated by Project Make:

MYLIB.DLL	External (/X) DLL needed to access Windows API subset.

Application Files Generated by Project Make

MYDIALOG.DLL	Contains dialog templates

APPENDIX D

PERFORMANCE ISSUES

D.1 Overview

The purpose of this Appendix is to provide suggestions on ways to manage performance when using MS-Windows.

D.2 RTIWIN.INI Options

Several of the options that can be included in the RTIWIN.INI file can be used to manage performance. Refer to Section 3.4 of this Addendum for details on the use of these options.

D.2.1 HaltRequestPeriod

Decreasing the value of this option increases the responsiveness of the NPL RunTime to the MS-Windows messages.

Increasing the value improves the performance by reducing the amount of time the NPL RunTime spends checking for MS-Windows messages, but may reduce responsiveness to operator actions.

This option is also used by the RunTime to determine how often to synchronize the screen display.

D.2.2 LockWaitTimeout and LockRetryDelay

These options can be used to improve the application's performance when a $OPEN is performed on a diskimage that has been opened by another workstation on a Novell NetWare network. Refer to Section 5.2.1 for more information.

D.2.3 ParallelFullDelay and ParallelRetryCount

These options can improve performance (speed up the printer) when using $OPEN to a local printer port.

D.2.4 ExclusiveWhenNetworkLocksHeld and Byte 43 of $OPTIONS.

These options can be used to control the amount of time tasks wait for files when switching to another window. Refer to Section 3.4 and Chapter 5 for details.

D.3 386 Enhanced Mode

386 Enhanced Mode is the recommended mode for executing MS-Windows. Use of standard mode may produce slower performance. 386 Enhanced Mode also allows the use of Virtual memory (Swap files) when using MS-Windows. Swap Files allows MS-Windows to use portions of the hard disk to swap out unused memory area when running an application. Refer to Section 3.2.5 for more information.

D.4 Memory

As a general rule, the more memory in the PC the better the performance of MS-Windows. If several MS-Windows applications are running simultaneously, the increased memory, if available, provides better performance.

The use of additional RAM provides substantially better performance than the use of Virtual Memory (Swap Files).