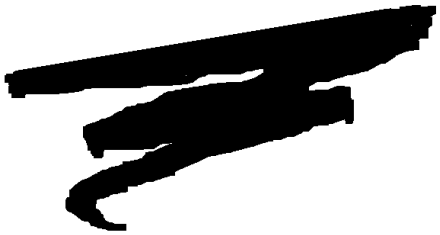
# NIAKWA PROGRAMMING LANGUAGE

# 386/DOS-EXTENDER 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 386/DOS-Extender is designed as an addition to the NPL Supplement for MS-DOS. This Addendum discusses the installation, operation, and 386/DOS-Extender specific features of the Niakwa 386 Interpreter and RunTime Program. For more information, refer to the appropriate NPL documentation and the Phar Lap 386/DOS-Extender documentation (if available).

## P.1 Prerequisite Knowledge

This guide assumes at least a basic knowledge of the IBM Personal Computer, the Microsoft Disk Operating System (DOS) Version 3.10 or greater.

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

# P.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 the 386/DOS-Extender.

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

Chapter 1 introduces the 386/DOS-Extender RunTime and Development Package diskette contents, and the specific features of the RunTime under the 386/DOS-Extender.

Chapter 2 discusses the installation procedures necessary for NPL under the 386/DOS-Extender.

Chapter 3 discusses the exact configuration requirements for NPL under the 386/DOS-Extender.

Chapter 4 discusses the RunTime operation under the 386/DOS-Extender.

Chapter 5 discusses the operating environment-specific language features under the 386/DOS-Extender.

Chapter 6 discusses the use of the External Call feature under the 386/DOS-Extender.

**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 main DOS Supplement.**

# CHAPTER 1

# INTRODUCTION

## 1.1 Overview

The NPL 386/DOS-Extender Addendum is intended as an aid in the correct installation and use of the 386/DOS Extender versions of the Niakwa Development Package and Run-Time programs (RTI386 and RTP386).

**NOTE: This addendum details the additional features of NPL operating under the 386/DOS-Extender environment. Refer to the MS-DOS Supplement for information on the standard NPL features.**

Section 1.2 describes the contents of the Niakwa 386/DOS-Extender Development Package.

Section 1.3 describes the contents of the Niakwa 386/DOS-Extender RunTime Package.

Section 1.4 discusses the specific features of the Niakwa 386/DOS Extender RunTime Program.

# 1.2 Contents of the Development Package

The NPL 386/DOS-Extender Development Package is intended for software development and execution of application software on MS-DOS or Novell NetWare systems using a 80386 processor or greater. The NPL Development Package for the 386/DOS-Extender is the same as for MS-DOS with the addition of the Niakwa 386/DOS-Extender Supplementary Files Diskette.

The contents of the standard NPL Development Package diskettes are listed in Section 1.2 of the MS-DOS Supplement. The following is a description of the additional files included on the 386/DOS-Extender Supplementary Files diskette.

| | |
|---|---|
| SWITCHES.DOC | This file documents the 386/DOS-Extender environment switches most likely to be used by NPL developers. Refer to Section 3.4 for more information. |

All other files contained in directories on this diskette are for use with the 386\DOS-Extender version of the NPL external call interface. Refer to Chapter 6 for more information.

# 1.3 Contents of the Runtime Package

**NOTE: The diskette labeled GOLD KEY is required to complete any task requiring or requesting the Gold Key diskette (i. e., installing the Gold Key security).**

The contents of the standard MS-DOS RunTime are listed in Section 1.3 of the NPL MS-DOS Supplement. The following is a description of the additional files included for the 386/DOS-Extender.

CFIG386.EXE This utility allows customization of startup switches that the Niakwa 386/DOS-Extender RunTime automatically uses every time the 386/DOS-Extender RunTime is executed.

RTI386.EXE The Niakwa 386/DOS-Extender Interpretive RunTime Program.

RTP386.EXE The Niakwa 386/DOS-Extender Non-interpretive RunTime Program.

RTIPERR.HLP This file is a text file that contains the Niakwa 386 RunTime error messages that are displayed optionally when using the 386/DOS-Extender Interpretive RunTime Program (RTI386).

RTIPERR.IDX This file contains the index listings used when the RTIPERR.HLP file is accessed.

# 1.4 386/DOS-Extender Runtime Specific Features

The 386/DOS-Extender version of NPL provides the following features in addition to those described in Chapter 1 of the MS-DOS Supplement:

- Extended memory support beyond 640K. Memory use is limited only by the physical memory available. The Niakwa 386/DOS-Extender RunTime, user partition, and external routines all reside in extended memory (above 1MB). Base memory may also be used based on switches set by the developer or user.

- Up to a 20% improvement on CPU operations.

- Support for variables larger than 64K. Refer to Section 5.4.

- Support for the standard Niakwa RunTime so that applications that are not compatible with the 386/DOS-Extender RunTime can still be run with the standard Niakwa RunTime Package.

**NOTE: This allows for any combination of the Niakwa 386/DOS-Extender RunTime and standard RunTime users up to the available user limit on network installations.**

- Support for virtual memory. Refer to Section 2.2 for more information on the necessary products.

- Support of mixed language programming through the use of the 386/DOS-Extender SDK. Refer to Chapter 6 for information on external calls use with the NPL 386/DOS-Extender RunTime Package. Refer to Section 2.2 for more information on the required products.

- Full support for the Niakwa 2227 Communication and Plot drivers.

- Support of all NPL MS-DOS and Novell NetWare features.

# CHAPTER 2

# INSTALLATION

## 2.1 Overview

This chapter provides instructions for installing the NPL 386/DOS-Extender Development and RunTime Packages.

Section 2.2 discusses the operating system, memory, and hardware requirements for the NPL 386/DOS-Extender RunTime.

Section 2.3 discusses the installation of the NPL 386/DOS-Extender Development Package.

Section 2.4 discusses the NPL 386/DOS-Extender RunTime security.

Section 2.5 discusses the installation of the NPL 386/DOS-Extender RunTime Package.

Section 2.6 discusses the installation of the Niakwa Gold Key security.

# 2.2 Operating System and Hardware Requirements

The 386/DOS-Extender version of NPL is designed to operate on systems meeting the following requirements:

- A minimum of a 80386-based (or higher) IBM-PC compatible.
- MS-DOS 3.1 or greater.
- Novell NetWare 2.10 or greater (for Novell Netware versions).
- A minimum of 2MB memory (1MB extended). 2MB provides approximately a 407K partition, more if base memory is also used (about 430K when RTI386 is configured with -MAXREAL 800H using DOS 5.0 loaded HIGH). All memory above 2MB is available to the user partition.
- Developers who wish to work with NPL external calls must license Phar Lap's 386/DOS-Extender SDK and 386/DOS-Extender Redistribution Package*, plus a supported 386 "C" compiler.
- Developers who wish to use virtual memory must license Phar Lap's 386 DOS-Extender SDK, 386/DOS-Extender Redistribution Package, 386/VMM Development version, and 386/VMM Redistribution Package*.

**NOTE:**
- Phar Lap products must be revision 5.0 or higher for Release IV compatibility.

* The above products are available from Phar Lap Software at:

**Phar Lap Software, Inc.**
60 Aberdeen Avenue
Cambridge, MA. 02138
Phone: (617) 661-1510
Fax (617) 876-2972

## 2.3 Installing the 386 Specific Development Software

Installation of the 386/DOS-Extender RunTime Package is identical with that of the standard RunTime except for the additional diskette(s). These diskette(s) should be copied to the same directory as the standard RunTime.

The 386/DOS-Extender Development Package Supplementary diskette contains one file (SWITCHES.DOC) which should be copied to the same directory where the standard Development Package is installed.

In addition, the 386/DOS-Extender Development Package Supplementary diskette contains a version of the BESDK External Call SDK for 386/DOS-Extender. Refer to Chapter 6 for installation instructions for the 386/DOS-Extender BESDK.

## 2.4 Security Issues

The 386/DOS-Extender RunTime utilizes the installed security (and Gold Key) from the standard RunTime that is included with this package. Install the standard Gold Key as described in Section 2.5 of the MS-DOS Supplement. All aspects of the Gold Key installation and security check work as described in the MS-DOS Supplement (or Novell NetWare Addendum for Novell NetWare installations).

**NOTE: The 386/DOS-Extender RunTime will operate only with the standard Gold Key shipped as part of the 386/DOS-Extender RunTime Package or with a Gold Key that has specifically been upgraded to support the 386/DOS-Extender. It will not work with any other Gold Key.**

# CHAPTER 3

# CONFIGURATION

## 3.1 Overview

Once the NPL 386/DOS-Extender Development and RunTime software has been installed, the 386/DOS-Extender environment must be configured to work with the Niakwa software. This procedure is examined in this chapter.

Section 3.2 discusses configuring the 386/DOS-Extender for use with NPL.

Section 3.3 discusses the CFIG386 Utility for customizing 386 RunTime operations.

Section 3.4 discusses setting the 386/DOS-Extender environment switches.

Section 3.5 discusses the various 386/DOS-Extender command line switches.

Section 3.6 discusses extended memory allocations.

Section 3.7 discusses linear memory allocations under virtual memory.

Section 3.8 discusses other 386/DOS-Extender environment switches.

Section 3.9 discusses using the 386/DOS-Extender switches with mixed language programming.

# 3.2 Configuring the 386/DOS-Extender Environment

This section is intended to aid in the configuration of the 386/DOS-Extender environment for NPL and NPL applications. All other configuration requirements remain the same as those documented in Section 2.3 of the MS-DOS Supplement.

## 3.2.1 Use of 386 Memory Managers

Use of extended memory managers, such as QEMM or HIMEM.SYS are allowed when using the NPL 386/DOS-Extender Runtime. However, these memory managers may use an amount of extended memory that may otherwise be used by the 386/DOS-Extender environment. Developers are encouraged to try different memory manager configurations and choose the settings best suited for their applications.

**NOTE: Memory managers typically work with the 386/DOS-Extender environment, but may require special setup or configuration. Refer to the documentation provided with the 386 memory manager for details.**

Certain TSR (Terminate and Stay Resident Programs) also may use an amount of extended memory that may otherwise be used by the 386/DOS-Extender environment.

# 3.3 The CFIG386 Utility

The CFIG386 Utility is provided for developers who intend to customize the 386/DOS-Extender environment for NPL applications.

**NOTE: This section only describes configuration options that are relevant to the 386/DOS-Extender environment and the NPL 386/DOS-Extender RunTime. The NPL 386/DOS-Extender Supplementary Files Diskettes contains a file called SWITCHES.DOC that contains additional information on other options.**

# 3.4 Setting the Environment Switches

There are two methods of setting the 386/DOS-Extender environment switches for the NPL 386/DOS-Extender RunTime:

- the CFIG386 Utility
- an MS-DOS environment variable.

## 3.4.1 Use of the CFIG386 Utility

To use the CFIG386 utility to add 386/DOS-Extender switches to the NPL 386/DOS-Extender RunTime, enter the following from a DOS prompt:

```
CFIG386 RTI386.EXE <NEW SWITCHES>
```

For example, to add the switch -MAXXMSMEM 100000h to the 386 RunTime program RTI386.EXE, enter:

```
CFIG386 RTI386.EXE -MAXXMSMEM 100000H
```

To view all switches currently configured in the NPL 386/DOS-Extender RunTime, enter:

```
CFIG386 RTI386.EXE
```

**NOTE: The CFIG386 Utility does not update the date or time stamp of the RunTime program (RTI386 or RTP386). It is necessary to use the method described above to verify that any new switch settings are added to the 386/DOS-Extender RunTime programs (RTI386 and RTP386).**

To clear all currently configured switches, enter the switch -CLEAR before the RTI386.EXE. For example, to clear all configured switches, enter:

```
CFIG386 -CLEAR RTI386.EXE
```

**NOTE: If a switch was set incorrectly, the -CLEAR parameter must be used before resetting the correct values of the respective switch.**

### 3.4.2 Setting Switches with an MS-DOS Environment Variable

Using a DOS environment variable is typically a better method to specify the 386/DOS-Extender environment switches.

To select an environment variable to contain the 386/DOS-Extender switches, configure the string "%varname" into the 386 RunTime Program (RTI386 or RTP386), where "varname" is the name of the environment variable.

**HINT:** A good choice for this environment variable is RTI386 or RTP386.

For example, to give users the ability to specify the 386/DOS-Extender environment switches in an environment variable called RTI386 enter:

```
CFIG386 RTI386.EXE %RTI386
```

The user can then specify the switch -MAXVCPIMEM 100000H by entering the MS-DOS command:

```
SET RTI386=-MAXVCPIMEM 100000H
```

**NOTE: It is recommended that this statement be added to the system's AUTOEXEC.BAT file. This can be accomplished by editing the batch file with a text editor (i.e., EDIT, EDLIN, etc.)**

If the specified environment variable does not exist, the NPL 386/DOS-Extender RunTime ignores it and runs with its preset switch settings.

## 3.5 Environment Command Line Switches

The switches described below are the 386/DOS-Extender environment switches that may be useful with the NPL 386/DOS-Extender RunTime. The default values assumed by the NPL 386/DOS-Extender RunTime are shown for each switch. If the default values are acceptable for the application, no changes are necessary when configuring the 386/DOS-Extender environment for the NPL 386/DOS-Extender RunTime.

**NOTE: Numbers entered as switch parameters are base 10 by default; they may be entered in hexadecimal (base 16) by appending the character "H" to the number (i.e., -MINREAL 4096 and -MINREAL 1000H are equivalent).**

### 3.5.1 Conventional Memory Allocations

#### -MAXREAL

Default Setting: FFFFH

Effect of default setting: The maximum amount of DOS memory that is reserved for DOS applications accessed using a $SHELL. The NPL 386/DOS-Extender RunTime uses extended memory exclusively.

Alternative Setting: 0

Effect of alternative setting: The minimum amount of DOS memory that is reserved for DOS applications accessed using a $SHELL. This may be insufficient memory for any $SHELL commands. The NPL 386/DOS-Extender RunTime uses both extended and conventional memory, resulting in a larger maximum work space size.

**NOTE: When -MAXREAL is set to 0, the partition size available is increased by the amount of base memory available. On typical configurations, this will be about 330K.**

#### -MINREAL

Default Setting: 0700H (may vary depending on release)

Effect of default setting: A minimum amount of DOS memory that must be available to allow the NPL 386/DOS-Extender RunTime to execute. This is necessary for the Niakwa security TSR program, and is required only at startup. This setting ensures that if the alternative setting **-MAXREAL 0** is used, sufficient memory is reserved to pass Niakwa's Gold Key security. Once the NPL 386/DOS-Extender RunTime is running, this real memory (28K) is released for use by DOS applications invoked by a $SHELL or for buffers (18K) required to access raw diskettes.

Alternative setting: Any value higher than the default setting.

Effect of alternative setting: A larger minimum amount of DOS memory is required to run the NPL 386/DOS-Extender RunTime. For applications that are known to require use of DOS applications accessed using a $SHELL, this setting ensures that the memory is reserved even if the alternative setting **-MAXREAL 0** is used.

## -MINIBUF

Default setting: (Set by the NPL 386/DOS-Extender RunTime)

Effect of default setting: Phar Lap allocates up to 16K of conventional memory for buffers used in file I/O and must have a minimum of 1K. Most I/O performed by the NPL 386/DOS-Extender RunTime is handled well by this default buffer allocation.

Alternative setting: A value greater than 16K and up to 64K

Effect of alternative setting: Applications that do significant amounts of I/O using buffers that can exceed 16K (COPY, MOVE DATALOAD BM or DATASAVE BM with large buffers) may benefit from using a larger buffer. Consequently, less conventional memory is available for other uses (i.e., $SHELL) or for the NPL 386/DOS-Extender RunTime if **-MAXREAL** is reduced.

## -MAXIBUF

Default setting: (Set by the NPL 386/DOS-Extender RunTime)

Effect of default setting: Phar Lap allocates up to 16K of conventional memory for buffers used in file I/O and must have a minimum of 1K. Most I/O performed by the NPL 386/DOS-Extender RunTime is handled well by this default buffer allocation.

Alternative setting: A value less than 16K down to 1K.

Effect of alternative setting: Applications that do little I/O using buffers that can exceed 1K may benefit from a smaller buffer. Consequently, more conventional memory is available for other uses(i.e., $SHELL) or for the NPL 386/DOS-Extender RunTime if **-MAXREAL** is reduced.

# 3.6 Extended Memory Allocations

When working in an environment where other applications may make use of extended memory while the NPL 386/DOS-Extender RunTime is executing, it may be necessary to change the configuration value of the **-MAXBLKXMS, -MAXXMSMEM, -MAXEXTMEM** or **-MAXVCPIMEM** switches to reserve extended memory for these applications. The default values permit the NPL 386/DOS-Extender RunTime to use as much extended memory as required, up to the full amount of available memory.

In environments in which applications make use of extended memory without using the XMS or VCPI standards, it may be necessary to set the **-EXTLOW** or **-EXTHIGH** parameters. This prohibits these applications from interfering with the operation of the NPL 386/DOS-Extender RunTime.

# 3.7 Linear Memory Allocations under Virtual Memory

Developers who wish to limit the amount of virtual memory used by the Phar Lap 386/DOS-Extender environment under VMM, should set the **-MAXPGMMEM** switch. There are several other VMM switches that can also be used to tune the performance of the virtual memory manager. Refer to the Phar Lap 386/VMM Reference Guide for more information.

# 3.8 Other Environment Switches

Other switches not described above are set appropriately by the NPL 386/DOS-Extender RunTime at startup. For specific settings and options for the other 386/DOS-Extender environment switches, refer to the appropriate Phar Lap manuals or the SWITCHES.DOC file contained on the 386/DOS-Extender Supplementary Files Diskette.

# 3.9 Use of the Switches with External Calls

Developers who intend to use the NPL 386/DOS-Extender RunTime to run applications linked with external routines in the 386/DOS-Extender environment (refer to Chapter 6), may require special settings to support unusual options in the external routines. This section briefly covers some of these options. Refer to the 386/DOS-Extender manuals for details of settings that may be necessary when using the NPL external call features in the 386/DOS-Extender environment.

### Privilege level

Default setting: -UNPRIVILEGED

Effect of default setting: The NPL 386/DOS-Extender RunTime is a well-behaved protected mode application that does not require ring 0 privilege. By running as an unprivileged application, the NPL 386/DOS-Extender RunTime permits control by the operating system over emulation, enhancing the ability of the product to run under emulation environments such as DPMI.

Alternative setting: **-PRIVILEGED**

Effect of alternative setting: Allows external routines to use ring 0 privilege.

## 3.9.1 Mixed Mode Operation

The NPL 386/DOS-Extender RunTime does not require special switch settings for mixed mode operation. However, developers that create a custom RunTime linked with external routines that require mixed mode (real and protected) operation, may need to set the -REALBREAK or **-CALLBUFS** arguments. Refer to the Phar Lap 386/DOS-Extender SDK documentation for more information.

# CHAPTER 4

# RUNTIME OPERATION

## 4.1 Overview

This chapter discusses the operation of the NPL 386/DOS-Extender RunTime.

Section 4.2 discusses the various 386/DOS-Extender environment virtual memory mode and its effect on the NPL 386/DOS-Extender RunTime.

Section 4.3 discusses starting the NPL 386/DOS-Extender RunTime.

Section 4.4 discusses the use of $SHELL with the NPL 386/DOS-Extender RunTime.

Section 4.5 discusses using the standard Niakwa RunTime.

## 4.2 386/DOS-Extender Virtual Memory Mode

The NPL 386/DOS-Extender RunTime can be executed with or without virtual memory support. Support of virtual memory requires developers to license Phar Lap's 386/DOS-Extender SDK, 386/DOS-Extender Redistribution, 386/VMM Development version, and 386/VMM Redistribution Package. Refer to Section 2.2 for more details.

## 4.3 Starting the Runtime

Procedures for starting the NPL 386/DOS-Extender RunTime is the same as documented in the NPL MS-DOS Supplement except as noted below:

- RTI386 and RTP386 should be substituted for RTI and RTP, respectively, in all startup commands.
- The 386/DOS-Extender environment must be properly configured as discussed in Chapter 3.

## 4.4 $SHELL

The behavior of $SHELL, under the NPL 386/DOS-Extender RunTime is similar to that under MS-DOS. However, the amount of memory available to $SHELL can be modified through the 386/DOS-Extender command line switches. By default base memory is not used by the RunTime, leaving this available for $SHELL. Refer to Chapter 3 for more information.

## 4.5 Using The Standard Niakwa Runtime

The standard Niakwa RunTime is also included with the NPL 386/DOS-Extender RunTime Package. There may be situations when the use of the standard version is necessary. For example, users of a Novell network who do not have extended memory or workstations with less than an 80386 based processor must use the standard RunTime version.

**NOTE: For instructions on using the standard Niakwa RunTime, refer to the MS-DOS Supplement.**

### 4.5.1 Serial Number

The serial number used by the standard Niakwa RunTime is the same as that used by the NPL 386/DOS-Extender RunTime. If the Gold Key security is installed on the hard drive, either RunTime can pass security from that hard drive. Otherwise, the RunTime prompts the user to insert the Gold Key diskette to pass security.

### 4.5.2 User Limit

On Novell Netware systems, the user count is increased by one each time either the standard or 386 version of the RunTime is started. The user count is the total number of users using the standard and the 386 version of the RunTimes.

### 4.5.3 Device Sharing

Device sharing, particularly non-network files and devices, is not allowed under the NPL 386/DOS-Extender RunTime.

Novell NetWare versions fully support all file sharing logic as described in the Novell NetWare Addendum.

# CHAPTER 5

# PLATFORM-SPECIFIC LANGUAGE FEATURES

## 5.1 Overview

This chapter discuses the platform-specific language features for the 386/DOS-Extender RunTime.

Section 5.2 discusses 386/DOS-Extender specific statements.

Section 5.3 discusses memory allocation under the 386/DOS-Extender RunTime.

# 5.2 Environment-Specific Statements

## 5.2.1 $MACHINE

Byte 1 of $MACHINE is set to "P" for the 386/DOS-Extender RunTime Version. All other $MACHINE values are the same as those for the standard Niakwa RunTime. Refer to Chapter 8 of the MS-DOS Supplement.

**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.**

## 5.2.2 $OPTIONS

There are no specific $OPTIONS bytes for NPL under the 386/DOS-Extender. All $OPTIONS values are the same as those for the standard Niakwa RunTime. Refer to Chapter 8 of the MS-DOS Supplement for details.

# 5.3 Memory Allocation

All NPL code and defined variables reside within a section of memory defined as the "user partition". With the NPL 386/DOS-Extender RunTime under the 386/DOS-Extender environment, the size of the user partition is limited only by the amount of extended memory in the computer on which the NPL 386/DOS-Extender RunTime is installed.

Due to the dynamic nature of memory allocation in the 386/DOS-Extender environment, the NPL 386/DOS-Extender RunTime allocates an initial 139K 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 NPL 386/DOS-Extender 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 NPL 386/DOS-Extender RunTime is unable to allocate the memory, the value of SPACEF then drops below 64K.

To illustrate this, consider the following:

| 386/DOS-Extender RunTime Environment | SPACEW Value | SPACEF Value |
|---|---|---|
| Initial RunTime Memory | 139664 | 139456 |
| Allocated 64K Variable | 139664 | 73920 |
| Allocate 8300 Bytes | 139664 | 65600 |
| Allocate 2nd 64K Variable | 205168 | 65600* |
| Allocate 20K Variable | 205168 | 45080** |

*Additional 64K segment allocated to user partition.

**Value SPACEF reports in the event the NPL 386/DOS-Extender RunTime was unable to allocate the extra 64K.

**NOTE: Once memory is allocated in this fashion, it is not released until the task is closed. Thus, in the example above, executing a CLEAR statement results in both SPACEW and SPACEF reports 205168.**

# 5.4 Support for Variables 64K

The 386/DOS-Extender RunTime is a true 386 implementation. Therefore, variables larger than 64K are fully supported as described in Chapter 4 of the NPL Programmer's Guide.

## 5.4.1 Memory Fragmentation

If a large variable is allocated, new space is requested from the operating system. When the variable is no longer required, the RunTime cannot return this memory to the operating system until a $END occurs. The allocation is still available to the NPL program, but any requests for substantially larger variables cannot use the space.

For example:

```
0010 ;PROGRAM1
   : DIM A$1000000                :;allocate 1MB from O/S
   : DIM B$1000000                :;allocate 1MB from O/S
   :;do some work...
   : PRINT SPACEW                 :;workspace size is about 2MB
   : LOAD T"PROGRAM2"

0010 ;PROGRAM2
   : DIM C$2000000                :;need 2MB from O/S
   : PRINT SPACEW                 :;workspace size is about 4MB
```

When the second program resolves, neither of the 1MB chunks requested from the operating system is big enough for the new variable, so new space must be allocated. The result is that a minimum workspace of 4MB is required. If only 3MB of extended memory were available, a memory error (A01) would occur.

**NOTE: If the programs were run in the opposite order (PROGRAM2, then PROGRAM1) both 1MB variables could be allocated from the 2MB chunk, and only 2MB of workspace would be necessary.**

There is a way to prevent this fragmentation problem, If you have some idea what the maximum workspace size is that you will need. Early in the application's startup, a sequence such as:

```
0010 DIM ReserveSpace$(0)1
   : MaximumWorkspaceNeeded=xxx  :;determine max size needed
   : ;Grow the workspace as one 'chunk'
   : MAT REDIM ReserveSpace$(1)MaximumWorkspaceNeeded
   : ERROR DO
   :     STOP "Insufficient memory to run"
   : ENDDO
   :;When the variable ReserveSpace$() is cleared (by an overlay)
   :;the allocated memory will be available as one chunk.
   : LOAD T"..."
```

can be used to ensure that the total workspace size used by RTI is effectively unfragmented.

Except for large variable allocations, the workspace normally expands in 64K "chunks".

# CHAPTER 6

# MIXED LANGUAGE PROGRAMMING

## 6.1 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 the 386/DOS Extender. For a complete discussion on the general operation of mixed language programming, refer to Chapter 11 of the NPL Programmer's Guide.

The remainder of this Section continues to provide an overview of the 386/DOS-Extender environment.

Section 6.2 discusses the contents of the 386/DOS-Extender BESDK.

Section 6.3 discusses the installation of the 386/DOS-Extender BESDK.

Section 6.4 discusses NPL external call support specific to the 386/DOS-Extender environment.

Section 6.5 discusses support of Metaware HIGH C under the 386/DOS-Extender.

Section 6.6 discusses support of 386 ASM Macro Assembler under the 386/DOS-Extender.

Section 6.7 discusses support of Metaware Professional Pascal under the 386/DOS-Extender.

Section 6.8 discusses "binding" the 386/DOS-Extender executables.

Section 6.9 discusses the memory allocation module requirements of a linked 386/DOS-Extender executable.

Section 6.10 discusses accessing real mode memory and TSR programs under the 386/DOS-Extender.

Section 6.11 discusses the flow control of external libraries.

**NOTE: The following chapter refers only to the 386/DOS-Extender BESDK package contained on the 386/DOS-Extender Supplementary Files Diskette. For information on the standard MS-DOS BESDK package, refer to Chapter 11 of the MS-DOS Supplement.**

### 6.1.1 Differences from DOS/SuperDOS Releases.

The MS-DOS and SuperDOS releases of the NPL RunTime use the quick library mechanism of the Microsoft Linker allowing the standard NPL RunTime program to load a specified set of routines after NPL starts. This approach does not work in the 386/DOS-Extender environment where code in general cannot be dynamically linked at execution time.

Instead, the user subroutines must be linked with NPL code itself, to produce an executable file which is a customized version of NPL with the user subroutines "built in". This is the approach used to support external subroutines under the 386/DOS-Extender environment.

### 6.1.2 Choosing the Development Environment

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

- Because the user subroutines must be linked to produce a 386/DOS-Extender binary executable file, the Phar Lap software development system utilities, particularly the linker ("386LINK") must be present. In addition, if the Metaware High C or Metaware Professional Pascal is being used, the Metaware High C or Professional Pascal startup and support libraries must be present.

- Working with many of the 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". To keep this documentation brief, the rest of the chapter refers only to NMAKE.

- To execute a 386/DOS-Extender binary (.EXP) file, each user must also be licensed to use the Phar Lap RUN386 program, must have a licensing agreement with Phar Lap that enables distribution of executables that have been bound with the bindable version of this product.

### 6.1.3 Security

Any custom version of the RunTime produced by using the BESDK procedures, although not physically copy protected, does not execute unless a 386/DOS-Extender enabled RunTime is installed on the system where the custom version is to be used.

**NOTE: The 386/DOS-Extender RunTime installed must be of the same type as the custom RunTime. That is, a 386/DOS-Extender RunTime must be installed. In addition, the revision level of the installed 386/DOS-Extender RunTime must be equal to or greater than the revision level of the custom RunTime.**

When executed, the custom RunTime extracts the Serial Number from the installed 386/DOS-Extender RunTime and uses the Serial Number to perform the security check. If the Gold Key security from the 386/DOS-Extender RunTime is installed on the system, the security check is passed based on that. Otherwise, the Gold Key security check is performed. If this is necessary, the Gold Key from the installed 386/DOS-Extender RunTime must be used.

**NOTE: The custom RunTime also extracts and enforces the user limit from the 386/DOS-Extender RunTime installed on the system.**

The procedure for installation of custom RunTimes at end user sites is simple:

1. Install a 386/DOS-Extender RunTime of the same version or greater as the custom RunTime on the end user's system using the normal installation procedure.

2. Copy the custom RunTime produced on the developer's development system to the end user's system.

### 6.1.4 Upgrades

When new releases of NPL become available, updates of the libraries used to make the customized versions of NPL will also be made available. Periodically, releases of the libraries with bug corrections corresponding to interim patches may also be made. It is the developer's responsibility to produce new versions of their customized RunTime programs and distribute them to their clients.

**NOTE: For formal upgrades of the RunTime, the 386 RunTime at end user sites must be upgraded before installing an upgraded custom RunTime.**

## 6.2 Contents of the 386/DOS-Extender BESDK

The BESDK files for the 386/DOS-Extender are stored on the 386/DOS-Extender Supplementary Files Diskette and are stored in a MS-DOS format. These files must be installed on the hard drive before they can be used. The INSTALLP 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 Metaware C example followed in the text.

The examples are provided mainly to test normal versions of the compilers, assemblers, linkers, etc., that are being used with the source files that have been pretested, and to help clarify any points that may be unclear in the text.

**HINT:** It is recommended that example standalone and customized versions of the BESDK examples be produced before starting a customized project, to ensure the various utilities work together as they should.

The contents of the 386/DOS-Extender BESDK are listed below:

| | |
|---|---|
| \ (root 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 the BESDK. |
| INSTALLP.BAT | A batch program to install the 386/DOS-Extender BESDK files to a specified directory and subdirectories. |
| **\BIN** | Contains two batch command files that can be used to link a set of external subroutines with either RTI386 or RTP386: |
| MAKERTIX.BAT | Links the specified object files with NPL to create an Interpretive 386/DOS-Extender RunTime with external subroutines (RTIX.EXP). |
| MAKERTPX.BAT | Links the specified object files with NPL to create a Non-interpretive 386 RunTime with external subroutines (RTPX.EXP). |

**NOTE: The above batch files are limited to specifying a maximum of 10 arguments (files and libraries) and the total length of all arguments may not exceed 128 characters (a MS-DOS limitation). However, arguments that are file lists may also be specified by specifying "@filename" as an argument.**

| | |
|---|---|
| **\INCLUDE** | This directory contains files that are common to all implementations or to all implementations of a specific language. It is recommended that no changes are made to the files in this directory. The files provided in this directory are: |
| MYBOOT.SRC | A NPL source file, which performs a simple test of the example external subroutine. |
| MYBOOT.OBJ | Compiled form of MYBOOT.SRC. |
| MYSTART.SRC | Source version of the NPL program used to test the example subroutines and FUNCTIONs. |
| MYMODULE.SRC | Source version of the NPL library module used to specify the interface to the example sub-routines and FUNCTIONs, and containing sample CALLBACK function. |

| | |
|---|---|
| MYMODULE.BS2 | Compiled versions 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 Metaware High C programs, with structure and type definitions. |
| RTPALL.INC | Include file for 386ASM programs, with structure and type definitions. |
| RTPALL.PI | Include file for Microsoft Pascal and the standard Niakwa program. |
| RTPALL.PPI | Include file for Metaware Pascal and the standard Niakwa program. |
| **\INCLUDE\D3X** | This directory contains files that are specific to the 386/DOS-Extender environment. It is recommended that no changes are made to the files in this directory. The files provided in this directory are: |
| RTPDEFFN.H | Operating system dependent macros to define the Metaware High-C language attributes of external routines. |
| RTPPARM.OBJ | Compiled version of RTPPARM.C using Metaware C. |
| RTPPARM.C | C subroutines to provide the rtpfn_getparminfo() function used to check function declarations. |
| MYMALLOC.C | A Metaware High C version of a memory support module. |
| MYMALLOC.ASM | A 386ASM version of a memory support module. |
| MYMALLOC.P | A Metaware Professional Pascal version of a memory support module. |

| | |
|---|---|
| MYINT.OBJ | Subroutines used to access the 386/DOS-Extender API and segmented addressable memory. |
| MYINT.ASM | The 386ASM source code for the above. |
| MYINT.INC | A 386ASM include file, specifying the structures used by myint.obj. |
| MYINT.H | A Metaware High C include file, specifying the structures used by myint.obj. |
| MYINT.PPI | A Metaware Professional Pascal include file, specifying the structures used by myint.obj. |
| **\LIB** | This directory contains the object files required to make customized NPL 386/DOS-Extender RTI and RTP programs in library form. Some specific .OBJ files and linker response files may also be located here. |
| **\D3XNOEXT** | This directory contains files for creating a 386/DOS-Extender RunTime without external subroutines; these are provided to allow a simple test of the linking procedure. No compiling or assembling is required. |
| NOEXTERN.OBJ | Object file containing 386ASM mainline and "RTPEXT" routines that have no external subroutines. |
| NOEXTERN.ASM | 386ASM source for the above object file (for information only, not required for test). |
| MYMALLOC.OBJ | Preassembled version of the 386ASM version of memory support routines. |
| MAKERTIX.BAT | Batch file to produce the customized RTI (RTIX.EXP) for the example. |
| MAKERTPX.BAT | Batch file to produce the customized RTP (RTPX.EXP) for the example. |

| | |
|---|---|
| MAKEFILE | Script for NMAKE utility to produce both customized RTI and customized RTP. |
| **\D3XCEXAM** | Contains example files for 386/DOS-Extender implementations using Metaware C. |
| **\D3XMEXAM** | Contains example files for 386/DOS-Extender implementations using 386ASM. |
| **\D3XPEXAM** | Contains example files for 386/DOS-Extender implementations using Metaware Professional Pascal. |

The above three directories include the following files:

| | |
|---|---|
| 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 Metaware High C programs |
| ASM | for 386ASM programs |
| P | for Metaware Professional Pascal programs |

| | |
|---|---|
| MAKEMAIN.BAT | Batch file to produce the mainline for the example. |
| MAKERTIX.BAT | Batch file to produce the customized 386/DOS-Extender RTI (RTIX.EXP) for the example. |
| MAKERTPX.BAT | Batch file to produce the customized 386/DOS-Extender RTP (RTPX.EXP) for the example. |
| MYMAIN.LNK | A linker response file that contains options used to make the mainline for the example. |

| | |
|---|---|
| MAKEFILE | Script for NMAKE utility to produce both mainline, customized 386/DOS-Extender RTI and customized 386/DOS-Extender RTP (all programs that are out of date are remade). To use, enter "nmake". |

The **\D3XCEXAM** directory also contains the following files, specifically for Release IV callback features:

| | |
|---|---|
| MYCALLBK.C | Source code to illustrate the use of a C function (mykeyin) that performs a callback to the NPL function 'Callback-Keyin |
| MYCALLBK.H | 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 specifications required by MYPROC.C |

**NOTE: If the makefile itself 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 HC386, 386ASM, 386LINK, 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 ..\D3X\INCLUDE).
- The command files "MAKERTIX.BAT" and "MAKERTPX.BAT" can be found in "..\BIN".

- 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 IPATH environment variable (Metaware Pascal and Metaware High C only).

The NMAKE script files assume the current (4.00.00 or later) version of the NPL compiler "B2C" is on the execution PATH (older versions of B2C may cause NMAKE to stop with a spurious exit code - in this case, rerun the NMAKE script).

# 6.3 Installation of the BESDK Diskette

The 386/DOS-Extender Supplementary Files Diskette has been produced in MS-DOS format on two different medias: 5-1/4" 1.2MB diskettes and 3-1/2" 720K diskettes.

To install the 386/DOS-Extender BESDK, insert the 386 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:INSTALLP A: .
```

To install from drive B:

```
B:INSTALLP B: .
```

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

# 6.4 386/DOS-Extender Support

The external call features of NPL are supported in 386/DOS-Extender environment as described in the following sections.

### 6.4.1 Environments

No special addressing models are used with the 386/DOS-Extender. All addresses are 32-bit pointers to the default code/data segment. Segmented addresses are not generally used by applications under the 386/DOS-Extender. Exceptions may be required to access MS-DOS memory, the MS-DOS environment block, or memory-mapped devices that are outside the default segment.

Operating system functions are accessed from the 386/DOS-Extender interface, which defines a set of functionality for interrupt 21H analogous in many ways to the MS-DOS interrupt 21H, extended to the 32-bit addressing mode. Developers may compile, link and run customized versions of 386/DOS-Extender RTI and RTP under the 386/DOS-Extender provided that access to the 386/DOS-Extender software development system, version 5.00 or later is present.

The entry point of the program depends on the language used, but must be specified by the external programs or library routines (the libraries and objects supplied do not define an entry point). Linking with the standard Metaware High C or Metaware Professional Pascal startup and support libraries meets this condition.

The external routines must also define a memory allocation subroutine for the 386/DOS-Extender RTP to use, to avoid conflicting memory allocation schemes in the various languages. Example routines are provided for each supported language.

Some special considerations may be necessary to take advantage of the optionally available virtual memory driver (VMMDRV.SYS).

## 6.5 Metaware HIGH C under the 386/DOS-Extender

Writing external subroutines in Metaware High C 386/DOS-Extender is relatively straightforward. Examples assume Metaware High C for MS-DOS 80386 environment, version 3.1 or later. In all the examples, the compiler chosen is the driver program (hc386). The Metaware compiler requires either the Phar Lap 386/DOS-Exender or a DPMI server such as Microsoft Windows to operate.

The Metaware High C compiler does not add an underscore ("_") to the start of all labels as some other compilers do. In the following discussion, labels are presented the way they must appear in the source files of your C routines.

### 6.5.1 General

Use the small (flat addressing) model 386 compiler on all "hc386" compile commands. This is the default for the compiler, so no options are required as part of the HCOPTS variable.

Make sure the include files provided with the BESDK are available either in a directory specified by a "-I"directory"" option to the "hc386" command or as part of the HCOPTS make variable. The directory should end in a backslash.

### 6.5.2 Mainline

The starting label of user code is called "main".

**NOTE: Substantial startup code from the Metaware High C library is executed before reaching the "main" routine. The standard entry point of an image linked using Metaware High C is "_MWINIT", a library routine.**

The 386/DOS-Extender RunTime subroutine should be referenced as an external with the standard BESDK calling conventions.

The default C language memory support routine (MYMALLOC.C) passes requests for memory from the 386/DOS-Extender RunTime routines to the "malloc" library routine.

### 6.5.3 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 386/DOS-Extender 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).

### 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 the 386/DOS-Extender defines "RTPDEFFN_EXT" as equivalent to this designation.

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

## 6.5.4 Linkage of Test Program

The files required for production of the standalone should include:

- The mainline
- The 386/DOS-Extender RunTime test subroutine. If callbacks to NPL are made by the test subroutines, it is usually practical to link a standalone test module which includes the code that makes these callbacks.
- The RTPEXT subroutine (optional, but recommended)
- The GOSUB' subroutines
- Any function or procedure subroutine. If these are used, the RTPPARM.OBJ module (located in \INCLUDE\DX3) must also be included.
- The Metaware High C support library HCE.LIB (name may vary)

## 6.5.5 Linkage of Customized 386/DOS-Extender RTI or RTP

The files required for production of the customized 386/DOS-Extender RTI or RTP should include:

- Files from the BESDK lib directory
- The mainline
- The RTPEXT subroutine

- The GOSUB' subroutines
- Any function or procedure subroutines. If these are used, the RTPPARM module (located in \INCLUDE\D3X) must also be used.
- The Metaware High C language memory support module (from BESDK)
- The Metaware High C support library HCE.LIB (name may vary)

The batch command files supplied on the BESDK diskette are the recommended way to specify the files needed by the customized 386/DOS-Extender RTP or RTI.

For example:

```
..\bin\makertpx mymain myrtpext mysub mymalloc myproc mycallbk rtpparm
                    -lib      hce386 hcsoft
```

produces "RTPX.EXP" (Non-interpretive 386/DOS-Extender RunTime, with extensions).

and

```
..\bin\makertix mymain myrtpext mysub mymalloc myproc mycallbk rtpparm
                    -lib      hce386 hcsoft
```

produces "RTIX.EXP" (Interpretive 386/DOS-Extender RunTime, with extensions). If the list of files which must be linked in to make the customized 386/DOS-Extender RTI and RTP exceeds 128 characters or 10 parameters (MS-DOS limitations) the list of files must be specified indirectly, as @filelist, where "filelist" is the name of a text file that contains a list of the required files.

### 6.5.6 Binding the Customized 386/DOS-Extender Executable RTI or RTP

Once the RTIX.EXP file is created, this file must be combined with the 386/DOS-Extender (and optionally 386/VMM) to produce a single executable file. This allows end-users to run the customized Runtime in protected mode without having to know that it is using the 386/DOS-Extender.

**NOTE: Because the 386/DOS-Extender is included as part of the application, after binding, there are no special installation procedures required and end users can run the customized RunTime just as the 386/DOS-Extender RunTime can be run.**

Refer to Section 6.8 of this Chapter for information on running the Phar Lap BIND386 utility.

# 6.6 386ASM Macro Assembler

External subroutines and the mainline can be written entirely in 386ASM Macro assembler if required. Macro assembler has the advantage that support code dragged in from libraries is usually minimal, and so the resulting library is often more compact than if written in a high-level language. However, the code is generally much more difficult to write and less portable when complete.

386ASM is normally included as part of the 386/DOS-Extender Software Development Kit. Examples assume Phar Lap's 386ASM version 3.0 or later. Earlier versions may also work but are not tested.

External libraries written in Macro Assembler do not support the FUNCTION, PROCEDURE interfaces or callbacks to NPL.

## 6.6.1 General

Make sure the include files provided with the BESDK are available in a directory specified by a "-I directory" option to the "386asm" command. The directory name should end in a backslash.

## 6.6.2 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 small-model conventions for segment names. Explicit segment names must be used. Be sure that:

- All code segments have combine class "CODE" and are part of the group named DGROUP.
- 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. References to specific segments using the SEG operator are not permitted in the 32-bit flat model code. The OFFSET operator should always be based using DGROUP to ensure a correct flat model address.

At entry SS:ESP are set to the system stack area. Do not attempt to use a different stack area.

If additional memory is required, be sure to use a routine that is compatible with the RTP_MALLOC routine defined for use by NPL.

The default 386ASM language memory support routine (mymalloc.asm) requires that the startup code call MYMALLOC_INIT, which trims the size of the standard CODE/DATA segment to a minimum. Requests to RTP_MALLOC extend the size of the standard segment, and return the address of the previously unmapped memory.

### 6.6.3 Calling Conventions for BESDK Subroutines

#### Test RTP Subroutines

Declare the 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 a near (the default) procedure with the name "RTPEXT". When called, the address of the RTPDEF structure (defined in the include file rtpall.inc) is on the stack as a 32-bit near (flat model) pointer. The first field of this structure is a RTPREQ structure (defined in the include file rtpall.inc).

#### GOSUB' Subroutines

Each subroutine should be defined as a near (the default) procedure. 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    OFFSET < string>          ;32-bit flat model address
PUSH    SIZE < string>            ;32-bit integer
```

**NOTE: The string size is an unsigned 16-bit integer, zero-extended to keep stack alignment on a DWORD address.**

A numeric parameter is passed as:

```
PUSH    OFFSET < rtpnum structure>  ;32-bit flat model address
```

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 + 8 bytes per string parameter).

### 6.6.4 Linkage of Test Program

Programs written in Macro Assembler must be linked to produce an executable file. All input files to "386LINK" 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 RTPEXT subroutine (optional, but recommended)
- The GOSUB' subroutines (i.e., MYSUB.OBJ)
- The 386ASM language memory support module (from the BESDK, i.e., MYMALLOC.OBJ)

The batch command file "MAKEMAIN.BAT" supplied on the BESDK diskette contains the recommended command to link the files needed by the test program, namely: 386LINK, MYMAIN.OBJ, MYRTP.OBJ, MYRTPEXT.OBJ, MYSUB.OBJ, MYMALLOC.OBJ, and @MYMAIN.LNK.

### 6.6.5 Linkage of Customized 386/DOS-Extender RTI or RTP

The files required for production of the customized RTI or RTP should include:

- Files from the BESDK \LIB directory
- The mainline (i.e., MYMAIN.OBJ)
- The RTPEXT subroutine (i.e., MYRTPEXT.OBJ)
- The GOSUB' subroutines (i.e., MYSUB.OBJ)
- The 386ASM language memory support module (from the BESDK)

The batch command files supplied on the BESDK diskette are the recommended way to specify the files needed by the customized 386 RTP or RTI.

For example:

```
..\bin\makertpx mymain.obj myrtpext.obj mysub.obj mymalloc.obj
```

produces "RTPX.EXP" (Non-interpretive 386/DOS-Extender RunTime, with extensions).

and

```
..\bin\makertix mymain.obj myrtpext.obj mysub.obj mymalloc.obj
```

produces "RTIX.EXP" (Interpretive 386/DOS-Extender RunTime, with extensions).

### 6.6.6 Binding the Customized 386 Executable RTI or RTP

Once the RTIX.EXP file is created, this file must be combined with the 386/DOS-Extender (and optionally 386/VMM) to produce a single executable file. This allows end users to execute the customized 386/DOS-Extender Runtime in protected mode without having to know that it is using the 386/DOS-Extender.

**NOTE: Because the 386/DOS-Extender is included as part of the application, after binding, there are no special installation procedures required and end users can run the customized RunTime just as the 386/DOS-Extender RunTime can be run.**

Refer to Section 6.8 of this Chapter for information on running the Phar Lap BIND386 utility.

# 6.7 Metaware Professional Pascal

Writing external subroutines in Metaware Professional Pascal for the 386/DOS-Extender is relatively straightforward. Examples assume Metaware Professional Pascal for MS-DOS 80386 environment, version 2.7 or later. Earlier versions may also work, but are not tested. In all the examples, the compiler chosen is the version for real-mode execution (pp386). Several extensions to standard Pascal are required to support the required functionality for the external calls. In particular, the standard packages "Loopholes" and "Other_languages" must be available to the example include files.

The Metaware Professional Pascal compiler does not add an underscore ("_") to the start of all labels as some other compilers do. In the following discussion, labels are presented the way they must appear in the source files of your Pascal routines.

External libraries written in Pascal do not support the FUNCTION, PROCEDURE interfaces or callbacks to NPL.

## 6.7.1 General

Use the small (flat addressing) model 386 compiler on all "pp386" compile commands. This is the default for the compiler, so no options are required as part of the PPOPTS variable.

Make sure the include files provided with the BESDK are available either in a directory specified by a "-I"directory"" option to the "pp386" command or as part of the PPOPTS variable. The directory should end in a backslash.

## 6.7.2 Mainline

Declare RTP as an external function with the standard BESDK calling conventions returning type CARDINAL result. It is not necessary to pass parameters to RTP in this case.

The starting label of user code is called "_MWMAIN". The "_MWMAIN" symbol is generated for any PP386 module that has a body associated with the "program" header.

**NOTE: Substantial startup code from the Metaware Professional Pascal library is executed before reaching the "_MWMAIN" routine. The standard entry point of an image linked using Metaware Professional Pascal is "_MWINIT", a library routine.**

The default Pascal language memory support routine (MYMALLOC.P) passes requests for memory from the 386/DOS-Extender Runtimes RTP routines to the "Malloc" library routine (part of the heap utility package).

### 6.7.3 Calling Conventions for BESDK Subroutines

#### Test RTP Subroutines

Declare 386 RTP as function named "RTP" returning cardinal.

Declare all GOSUB' routines using standard BESDK calling conventions (i.e., the sub-routine preserves non-volatile registers, arguments are pushed in the order they appear, arguments are popped by the called routine). Argument types are "x:rtpstr_pointer, len:cardinal" for a string and "x:rtpnum_pointer" for a numeric.

Call GOSUB' subroutines with arguments in format "Adr(x[1])%retype rtpstr_pointer,length(x)" for a string, and "Adr(x)%retype rtpnum_pointer" for a numeric.

Precede all GOSUB' routine declarations using the following pragma:

```
pragma Calling_convention([callee_pops_stack]);
```

Follow all GOSUB' routine declarations using the pragma:

```
pragma Calling_convention();
```

#### RTPEXT Subroutine

Declare RTPEXT as a function named "RTPEXT" returning an integer.

Assign the address of the GOSUB' procedure to rtpdef_pointer using the Address(x) function.

Assign the address of the LIST' description using the retype(address(x[1]),rtpstr_pointer) function. Strings used for this purpose should not be declared as local variables to RTPEXT, since this places them in a volatile area (the stack).

The RTPEXT subroutine should be defined as a procedure named "RTPEXT" with the standard BESDK calling conventions. When called, the address of the rtpdef structure (defined in the include file RTPALL.PPI) is the only parameter. The first field of this structure is a rtpreq structure (also defined in the include file RTPALL.PPI).

### GOSUB' Subroutines

Subroutines called by the GOSUB' interface should be declared as functions returning type integer results. Since they are usually in a source file separate from the RTPEXT subroutine and require linking, Metaware Professional Pascal requires a duplicate function declaration with the "external" designation replacing the body.

The data type of the NPL strings of different lengths are usually incompatible types in standard Pascal, and extensions to support variable-length strings require a different format and parameter passing convention from that used by NPL. The approach used by the example pascal subroutines is to declare a "rtpstr" type that is the largest array of characters, with indices starting at 1. This means that subscript checking on these strings is not enforced within the pascal subroutines, and caution must be taken to ensure that string bounds are not exceeded.

Precede all GOSUB' routine declarations using the following pragma:

```
pragma Calling_convention([callee_pops_stack]);
```

Follow all GOSUB' routine declarations using the pragma:

```
pragma Calling_convention();
```

Formats of strings in NPL do not have appropriate format for use by Pascal. If strings are to be used by Pascal library routines you must make copies that have trailing spaces removed and appropriate length definitions.

## 6.7.4 Linkage of Test Program

Programs written in Metaware Professional Pascal must be specifically linked to produce a 386/DOS-Extender executable file. All input files to 386LINK must be the result of previously run compiles or libraries of files.

The files required for production of the standalone should include:

- The mainline
- The RTP () test subroutine

- The RTPEXT subroutine (optional, but recommended)
- The GOSUB' subroutines
- The Metaware Professional Pascal support library ppe.lib (name may vary)

### 6.7.5 Linkage of Customized 386/DOS-Extender RTI or RTP

The files required for production of the customized 386/DOS-Extender RTI or RTP should include:

- Files from the BESDK lib directory
- The mainline
- The RTPEXT subroutine
- The GOSUB' subroutines
- The Professional Pascal language memory support module (from BESDK)
- The Professional Pascal support library ppe.lib (name may vary)

The batch command files supplied on the BESDK diskette are the recommended way to specify the files needed by the customized 386/DOS-Extender RTP or RTI.

For example:

```
..\bin\makertpx mymain.obj myrtpext.obj mysub.obj mymalloc.obj ppe.lib
```

produces "RTPX.EXP" (Non-interpretive 386/DOS-Extender RunTime, with extensions), and

```
..\bin\makertix mymain.obj myrtpext.obj mysub.obj mymalloc.obj ppe.lib
```

produces "RTIX.EXP" (Interpretive 386/DOS-Extender RunTime, with extensions). If the list of files that must be linked in to make the customized 386/DOS-Extender RTI and RTP exceeds 128 characters or 10 parameters (MS-DOS limitations) the list of files must be specified indirectly, as @filelist, where "filelist" is the name of a text file that contains a list of the required files.

### 6.7.6 Binding the Customized 386/DOS-Extender Executable RTI or RTP

Once the RTIX.EXP file is created, this file must be combined with the 386/DOS-Extender (and optionally 386/VMM) to produce a single executable file. This allows end users to run the customized Runtime in protected mode without having to know that it is using the 386/DOS-Extender.

**NOTE: Because the 386/DOS-Extender is included as part of the application, after binding, there are no special installation procedures required and end users can run the customized RunTime just as the 386/DOS-Extender RunTime can be run.**

Refer to Section 6.8 of this Chapter for information on running the Phar Lap BIND386 utility.

## 6.8 Binding

To bind the RTIX.EXP file with the 386/DOS-Extender (or optionally 386/VMM), the Phar Lap BIND386 utility is required from the 386/DOS-Extender SDK (refer to the Phar Lap SDK BIND386 Utility Guide). The customized RTIX.EXP file can be bound using the BIND386 utility as shown below.

```
BIND386 \PHARLAP\RUN386 rtix.exp
```

Where the 386/DOS-Extender image RUN386.EXE from the \PHARLAP directory is bound with the customized RTIX.EXP protected mode application image in the current directory, creating the output customized 386/DOS-Extender RunTime program RTIX.EXE (the same technique can be used to produce an executable RTPX.EXE).

**NOTE: The command for binding with VMM support is slightly different. Refer to the Phar Lap VMM documentation for details.**

Other options for the BIND386 are available and can be found documented in the Phar Lap BIND386 Utility Guide.

# 6.9 Memory Allocation Module Requirement

Unlike other environments that use linkable libraries to make custom 386/DOS-Extender RTI and RTP programs (i.e., XENIX/UNIX), the method used to allocate task memory to the 386/DOS-Extender applications varies depending on the language model in use. Consequently when linking a custom 386/DOS-Extender RTI or RTP, part of the external routines requirement is to provide a subroutine (RTP_MALLOC) which provides the address of new memory areas to RTP in a way that does not conflict with the language of choice.

The interface to the subroutine RTP_MALLOC is roughly equivalent to the familiar C malloc() function. It is passed a (long) integer and returns the (32-bit flat model) address of a memory area in the default code / data segment that is for the exclusive use of the RTP. If for any reason memory is not available, a null pointer (value 0) must be returned.

The example programs include source code for RTP_MALLOC routines suitable for use with each of the example languages. These are located in the \INCLUDE\D3X directory, under the name MYMALLOC.x (x= C, ASM or P).

**NOTE: The customized 386/DOS-Extender RTI or RTP need not use any of these exact routines, but must provide an RTP_MALLOC routine, and whatever routine is used must be compatible with the language is being used for the external routines.**

## 6.9.1 386ASM Version of RTP_MALLOC

The method of memory allocation for 386ASM is to reduce the size of the default segment to the minimum permitted at startup time, and extend this limit when requested (in units of 4K) using an available 386/DOS-Extender call.

**NOTE: This method requires correct estimation of stack requirements. If the routines use substantial amounts of stack space (due to recursion or large dynamically allocated variables), increase the amount of space allocated to stack. This method automatically uses virtual memory if the VMM driver is enabled.**

### 6.9.2 Metaware High C Version of RTP_MALLOC

The method of memory allocation for Metaware High C programs uses the malloc() function of the standard library. The behavior of this function is implementation dependent. Under the current implementation this method automatically uses virtual memory if the VMM driver is enabled.

### 6.9.3 Metaware Professional Pascal Version of RTP_MALLOC

The method of memory allocation for Professional Pascal programs uses the Malloc() function from the "heap" standard Utility package. The behavior of this function is implementation dependent. Under the current implementation this method does not automatically use virtual memory if the VMM driver is enabled.

## 6.10 Accessing Real Mode Memory and TSR Programs

The 386/DOS-Extender provides a number of facilities to allow programs executing in the 32-bit protected model to access memory and programs that are stored in DOS real-mode memory (which is normally located in a segment different from that used by the 32-bit small-model address space). These facilities are most easily accessed using assembly language from the MS-DOS system call interrupt (21H). Refer to the Phar Lap386/DOS-Extender Reference Manual for details of the API.

Access to the 386/DOS-Extender API from Metaware High C is provided by use of the intdos() and intdosx() functions. Access to the 386/DOS-Extender API from Metaware Professional Pascal is provided by use of the "msdos.pf" utility package. Refer to the Metaware documentation for details.

**NOTE: In each case, the names used for registers (ax, bx, etc.) are nominally the names of 16-bit registers, but under the 386/DOS-Extender these are in effect the corresponding 32-bit register (eax, ebx, etc.).**

Access to data outside the default segment is possible under Metaware High C by use of far pointers, and under Metaware Professional Pascal by the use of the Longptr type. Refer to the Metaware documentation for details.

Alternative support for access to the 386/DOS-Extender API from Metaware High C and Metaware Professional Pascal is provided by BESDK's MYINT.OBJ module. The MYINT and MYMOVSB routines in this module accept the same single parameter. This parameter is the address of a structure that specifies the contents of registers before the operation and which receives the contents of the registers after the operation.

**NOTE: Valid values or 0 must be specified for all segment registers (DS, ES, FS, GS), whether these registers are required for the operation or not. Failure to ensure that this is done results in an address fault and termination of the application. The fields in this structure are documented in the include files MYINT.x (x= H, INC, OR PPI depending in the language in use).**

The MYINT21 routine permits calling interrupt 21H with all registers specified and allows inspection of results which return in registers. As a convenience, the return value of this function is 0 if the carry flag was clear after the call, or -1 if the carry flag was set. A set carry flag is often used to indicate an error condition. The value of the carry flags when MYINT21 is called is ignored. Only after the call is made are the resulting flags stored in the flags status variable.

The MYMOVSB function allows performance of an inter-segment data transfer (from DS:ESI to ES:EDI) for a specified number (ECX) of bytes.

The MYDSSEG function returns the value of the current default segment, which may be required for some 386/DOS-Extender operations.

When a 386/DOS-Extender program needs to call real-mode TSR's, the following steps are required:

1. Identify any parameters to the TSR. If all parameters are in registers, and these are not addresses of other parameter blocks, in general the TSR may be called directly. Refer to step 3.

2. For any parameter blocks required by the TSR, use the 386/DOS-Extender call 250FH to determine whether the parameter block has an equivalent real mode address. Both the protected mode address and the length of the parameter block must be specified. If this call returns a result indicating that the parameter block is not in the MS-DOS address space, a MS-DOS memory block must be allocated using the 386/DOS-Extender call 25C0H and the parameter block copied to the new location (using MYMOVSB subroutine) before the call is made. Once allocated, the MS-DOS memory area at segment X can be accessed in protected mode (by MYMOVSB) at segment 34H offset X*10H.

3. Use the 386/DOS-Extender functions 250EH, 2510H or 2511H (depending on whether the TSR is called as a subroutine or as an interrupt) to call the real-mode TSR. Calls 2510H and 2511H require that a parameter block be passed that specifies the real-mode values of segment registers and registers used by the 386/DOS-Extender call. Do not confuse these with the MYINT structure.

4. If real mode copies of parameter blocks were made before the call, copy the real mode versions onto the protected mode versions (using MYMOVSB) and free the MS-DOS memory blocks using the 386/DOS-Extender call 25C1H.

5. If the TSR returns information in the form of a pointer to a real-mode data area (which was not a parameter), copying this to a protected mode addressable area may be necessary.

Some optimization of the above procedure may be possible for specific calls (to reduce the number of times allocation and deallocation routines are called) if the size of parameter blocks is known in advance. In addition, the above assumes that the addresses passed to TSR's are transient, i.e., the TSR's will not assume these addresses continue to remain valid after the call returns.

If parameter blocks themselves contain pointers to other parameter blocks, care must be taken to ensure that all pointers are correct (real mode addresses) before copies are made to the real address space.

**NOTE: In general, making a call to a real-mode procedure requires some careful analysis to make sure that all pointers referenced by the call have correct real mode equivalents. It also requires knowledge of the length of any parameter areas passed to (or returned by) the TSR.**

# 6.11 Flow Control for External Subroutines

The flow control (in chronological order) for 386/DOS-Extender RTI or RTP using external subroutines is as follows:

1. C or Pascal startup routines execute and perform some initialization, and eventually call the external library mainline (i.e., main()).

2. The mainline performs some initialization work for the external subroutines, and calls RTP(). If assembly language is used, initialization of the memory allocation support module is required prior to calling RTP.

3. NPL runs and does some initial configuration work, including processing command-line options and loading the bootstrap program.

4. NPL scans the external library for numbered DEFFN's with named aliases, using the LIST' calls starting at function number 0. An internal table of identifiers and equivalent externals is built.

5. NPL execution proceeds. At some point, a GOSUB', (e.g., GOSUB'100) is executed, and no local GOSUB' subroutine is found. If the GOSUB' is to a named DEFFN', and the identifier is found in the table created in step 4, the equivalent number is used to query RTPEXT.

6. NPL calls RTPEXT to find out whether an external '100 subroutine exists, and if so, where it is and what parameter types it needs.

7. RTPEXT supplies the requested information (i.e., GOSUB'100 exists, has 3 parameters with types string, string, and numeric, which is located at mysub()) and returns (to NPL).

8. If the RTPEXT indicated that the subroutine does not exist, or if the number and type of parameters do not match, a NPL error is generated on the GOSUB' statement. Otherwise, NPL evaluates parameters and calls the external subroutine (mysub) whose address was provided by RTPEXT.

9. The external subroutine (mysub) executes and returns to NPL. NPL execution proceeds until we are back at step 5 (another GOSUB') or the rtp is ending ($END, Killed from HELP, etc.). In the second case, go to the next step.

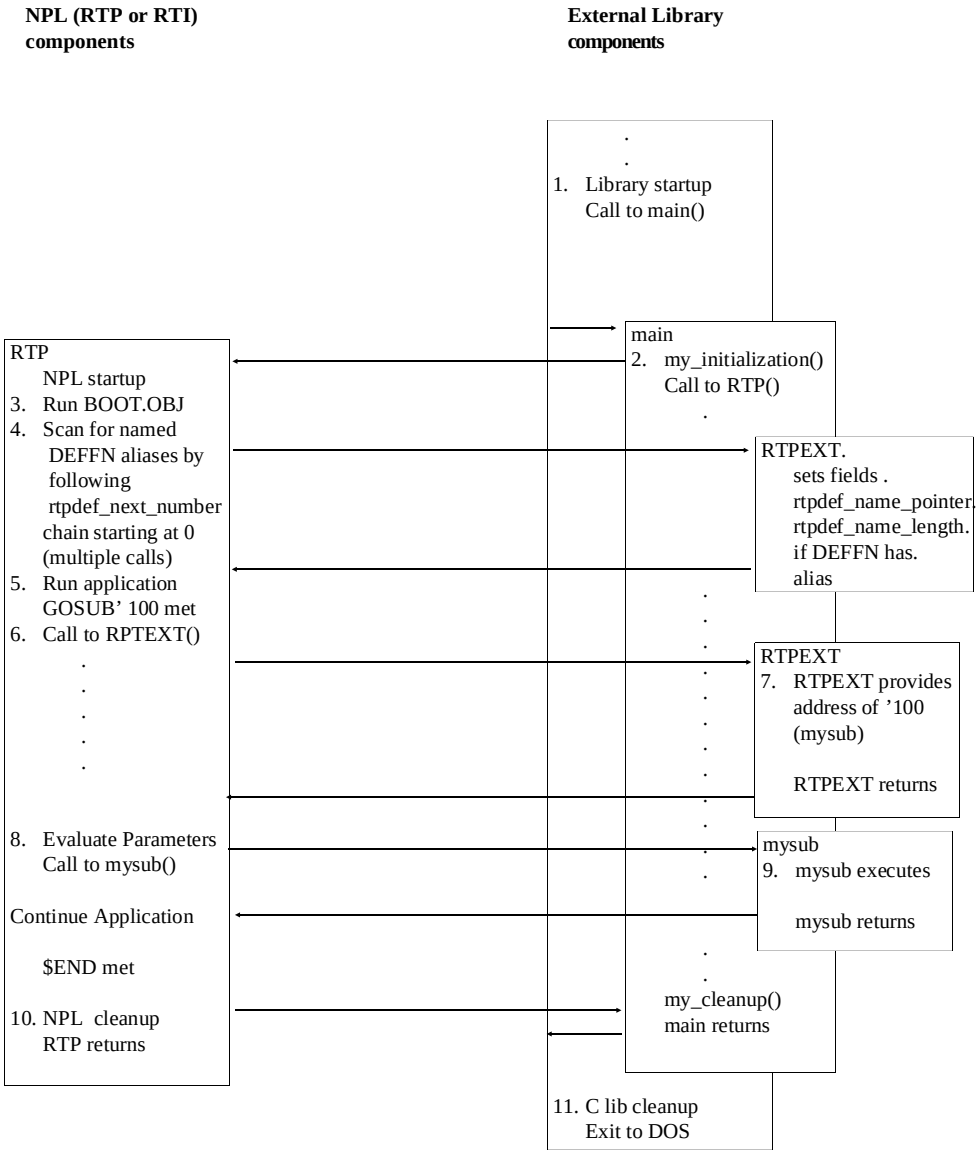
10. NPL does its cleanup, then the RTP () subroutine returns to the caller (in the external library mainline).

11. The external library mainline does C or Pascal library shutdown, and eventually exits back to MS-DOS.

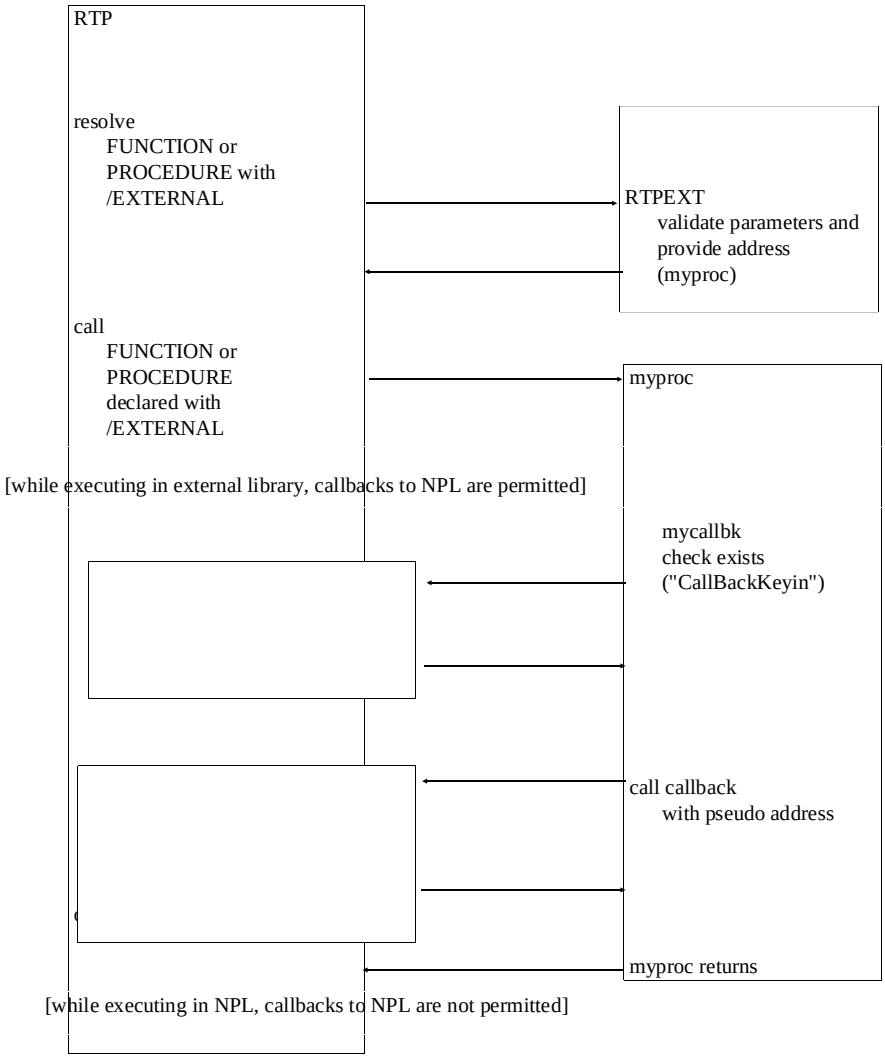
**NOTE: RTPEXT can be called by LIST' to find out information about DEFFN' subroutines without actually calling the subroutines.**

**Subroutines should not DEPEND on the above flow control order to work (i.e., a subroutine should not expect RTPEXT to always be called immediately before it).**

**The above outline is provided merely as a guide to your understanding of how the external mainline, rtp, RTPEXT and external subroutines interact.he following diagram shows how execution proceeds using the various software components with the above steps labeled.**


NPL (RTP or RTI) components
External Library components
.
.
1. Library startup
Call to main()
main
2. my_initialization()
Call to RTP()
.
RTP
NPL startup
3. Run BOOT.OBJ
4. Scan for named DEFFN aliases by following rtpdef_next_number chain starting at 0 (multiple calls)
5. Run application GOSUB' 100 met
6. Call to RPTEXT()
.
.
.
.
.
RTPEXT.
sets fields .
rtpdef_name_pointer.
rtpdef_name_length.
if DEFFN has.
alias
RTPEXT
7. RTPEXT provides address of '100 (mysub)
RTPEXT returns
8. Evaluate Parameters
Call to mysub()
mysub
9. mysub executes
mysub returns
Continue Application
$END met
10. NPL cleanup
RTP returns
my_cleanup()
main returns
11. C lib cleanup
Exit to DOS

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



# TABLE OF CONTENTS

## PREFACE



## INTRODUCTION



## INSTALLATION



## CONFIGURATION

## RUNTIME OPERATION



## PLATFORM-SPECIFIC LANGUAGE FEATURES



## MIXED LANGUAGE PROGRAMMING