North American Response Center

HP 3000 APPLICATION NOTE #34

RIN Management

(Using COBOLII Examples)

HEWLETT PACKARD

September 1, 1987
Document P/N 5958-5824R2736

RESPONSE CENTER APPLICATION NOTES

HP 3000 APPLICATION NOTES *are published by the North American Response Center twice a month and are distributed with the Software Status Bulletin. These notes address topics where the volume of calls received at the Center indicates a need for addition to or consolidation of information available through HP support services.*

Following this publication you will find a list of previously published notes and a Reader Comment Sheet. You may use the Reader Comment Sheet to comment on the note, suggest improvements or future topics, or to order back issues. We encourage you to return this form; we'd like to hear from you.

RIN Management

INTRODUCTION

The purpose of this Application Note is to give experienced programmers a guide to resource management on the HP 3000.

A *resource* is any source of information that at some time may be required by a process. When such resources are shared with other processes, resource management is important because each process typically requires temporary, exclusive access to the resources to ensure consistency. In general, resource management is accomplished by resource locking.

MPE provides four different resource locking mechanisms:

- File system locking, for flat files and KSAM files.
- Database locking, for IMAGE databases, data sets, and data entries.
- Global Resource Identification Numbers (Global RINs).
- Local Resource Identification Numbers (Local RINs).

This note addresses Global and Local RINs. For further information on file system and database locking, see the *Additional Reading* section at the end of this Note.

GLOBAL RINs can be used by processes in different jobs and remain in existence until deleted. Local RINs are acquired by a job (which may include several processes) and are known only to that job. They are released either explicitly or upon return to the job's Command Interpreter (C.I.).

The user of a RIN decides what the RIN 'protects'. Locking RINs is an agreement between the processes using the RINs just as a 'locking scheme' must be developed when accessing IMAGE databases. All processes *must* agree on which RINs protect which resources in order for the RIN locking mechanism to be effective.

GLOBAL RINS

Global RINs are used to control access to resources shared by processes in different jobs and/or sessions. Before they can be used, they must be acquired by using the MPE command `:GETRIN` and by providing passwords which will grant access to them. For example:

```
:GETRIN RINPASS1
```

In this example `RINPASS1` is the password for the RIN number returned by MPE. How is the RIN number assigned? MPE will search the RIN Table for an available entry, display its number, (for example, '33'), and assign `RINPASS1` as its password. Another `:GETRIN` command might then assign 34 but there's no guarantee of this. Note that this command only makes the RIN available for use.

To release a RIN, use the MPE command `:FREERIN`. For example:

```
:FREERIN 33
```

If no users are holding locks against this RIN it is released and returned to the RIN Table. The *only* user who may release a RIN is the user who originally acquired it (that is, the one who issued the :GETRIN command).

WARNING

If a RIN is released there is no guaranteed way of acquiring the same one again. Since the simplest way to reference Global RINs is to 'hard code' them into programs, the loss of a RIN may mean that programs referencing them will need to be modified.

Global RIN Intrinsics

MPE provides two intrinsics for the management of Global RINs:

LOCKGLORIN	Requests either an unconditional or conditional lock on a specified Global RIN.
UNLOCKGLORIN	Unlocks a previously locked Global RIN.

You will notice that there are no intrinsics to acquire or release Global RINs; you *must* use the MPE commands previously mentioned. You could execute these commands from within a program by using the the COMMAND intrinsic; however, the number returned by :GETRIN is displayed on the $STDLIST device. This would be useless if the program also needed to lock and unlock this newly acquired RIN!

For example:

```
01  GLOBAL-RIN-DATA.
    05  GLOBAL-RIN-NUM          PIC S9(4) COMP VALUE 33.
    05  RIN-LOCK-CONDITION      PIC S9(4) COMP VALUE 1.
    05  GLOBAL-RIN-PASS         PIC X(8)  VALUE "RINPASS1".
```

In this example, note that the RIN number, 33, and the password, RINPASS1, have been hard coded in the program. In addition, RIN-LOCK-CONDITION is set to 1 indicating that UNCONDITIONAL locking is be used. A value of ZERO indicates CONDITIONAL locking. Both of these techniques will be discussed.

Locking and Unlocking Global RINs

To lock a Global RIN:

```
CALL INTRINSIC "LOCKGLORIN" USING GLOBAL-RIN-NUM,
                                  RIN-LOCK-CONDITION,
                                  GLOBAL-RIN-PASS.
IF C-C NOT = 0 THEN
     .
     .
     .
```

It is important to check the condition code returned by the lock attempt.

If the lock is successful, a C-C = 0 condition ('CCE') is returned, and you 'own' that RIN until you call UNLOCKGLORIN:

```
CALL INTRINSIC "UNLOCKGLORIN" USING GLOBAL-RIN-NUM.
IF C-C NOT = 0
     .
     .
     .
```

Again, even when unlocking, it is wise to check the condition code to ensure that the call was successful.

LOCAL RINS

As stated previously, Local RINs are Resource Identification Numbers known only within the job which requests them. They are easier to use than Global RINs and, unlike Global RINs, do not have any MPE commands to acquire and release them. Rather, this is accomplished entirely within the program or programs which use them, using the following intrinsics:

GETLOCRIN	Requests a specified number of Local RINs be allocated for the process or process tree.
LOCKLOCRIN	Requests that a Local RIN be locked, either conditionally or unconditionally.
LOCRINOWNER	Returns the Process Identification Number (PIN) of the process that has the specified Local RIN locked.
UNLOCKLOCRIN	Requests that a Local RIN be unlocked.
FREELOCRIN	Releases all previously acquired Local RINs.

Local RIN Applications

Local RINs only have application in environments where more than one process has been created (known as 'FATHER' and 'SON' processes). Local RINs are used in this situation when the various processes in the structure need to share resources or require synchronization where the locking and unlocking of a Local RIN is used as the signaling mechanism or 'semaphore'. For this discussion refer to the following excerpt from a COBOLII program:

```
01  LOCAL-RIN-DATA.
    05  NUM-RINS-REQUESTED     PIC S9(4) COMP VALUE 10.
    05  CURRENT-RIN            PIC S9(4) COMP VALUE 0.
    05  LOCK-CONDITION         PIC S9(4) COMP VALUE 1.
    05  LOCAL-RIN-OWNER        PIC S9(4) COMP VALUE 0.
```

Allocating Local RINs

Local RINs are allocated once for all processes currently executing below the job/session Command Interpreter. (Upon return to the C.I., MPE will release all such RINs automatically.) That is, only one process actually calls GETLOCRIN but it must allocate all the RINs that will be needed for all processes

using them. Note: the processes that are to use them do not have to be created prior to the call to GETLOCRIN.

If additional RINs are needed, a call to FREELOCRIN must be made to first release all RINs currently allocated before a new call to GETLOCRIN is made. See the discussion of FREELOCRIN below.

To acquire the needed Local RINs the process calls the GETLOCRIN intrinsic. For example:

```
CALL INTRINSIC "GETLOCRIN" USING NUM-RINS-REQUESTED.
IF C-C NOT = 0 THEN
     .
     .
     .
```

You should always check the condition code after a call to GETLOCRIN. If MPE cannot satisfy the entire request it will not allocate *any* RINs at all.

Locking and Unlocking Local RINs

The methods for locking Local RINs is essentially identical to the method used for locking Global RINs except that there is no 'RIN password' for Local RINs.

If a process wishes to lock a Local RIN it calls LOCKLOCRIN. For example:

```
CALL INTRINSIC "LOCKLOCRIN" USING CURRENT-RIN,
                                  LOCK-CONDITION.
IF C-C NOT = 0
     .
     .
     .
```

The value of 'CURRENT-RIN' can be any integer from 1 to the value of RINS-REQUESTED that was passed to the GETLOCRIN intrinsic. If, for example, Local RIN 3 were locked then CURRENT-RIN might contain a value of 3.

As with Global RINs, 'LOCK-CONDITION' can be a 1 to indicate an unconditional attempt to lock, or 0 denoting a conditional attempt.

Unlocking a Local RIN is accomplished by calling UNLOCKLOCRIN and passing the number of the Local RIN to be unlocked. For example:

```
CALL INTRINSIC UNLOCKLOCRIN USING CURRENT-RIN.
IF C-C NOT = 0 THEN
     .
     .
     .
```

In addition to UNLOCKLOCRIN, the SUSPEND intrinsic can be used to unlock a local RIN prior to suspending the caller. This might be used in applications where several processes are synchronized using local RINs. For example, a process ACTIVATEs its son; the son immediately attempts to lock a local RIN. The 'father' process SUSPENDs itself and releases the RIN at the same time.

```
CALL INTRINSIC "ACTIVATE" USING SON-PIN, 0.
IF C-C = 0 THEN
    CALL INTRINSIC "SUSPEND" USING 1, CURRENT-RIN.
```

In this example SON-PIN contains the PIN returned from CREATE. ACTIVATE is called using this value and a zero to indicate that it should not be suspended yet. It then calls SUSPEND to do this (the '1' indicates it expects to be ACTIVATEd by one of its sons) and it releases the lock held on CURRENT-RIN.

Determining Who Has a Local RIN Locked

LOCRINOWNER can be used to determine which process has a specified RIN locked. This intrinsic returns the Process Identification Number (PIN) of the process which has the RIN locked. For example:

```
CALL INTRINSIC "LOCRINOWNER" USING CURRENT-RIN
                             GIVING LOCAL-RIN-OWNER.
IF C-C NOT = 0 THEN
     .
     .
     .
```

The intrinsic returns a CCG or C-C > 0 condition if the specified RIN is not locked by any process. Otherwise, it returns the PIN of the process which has the specified RIN locked into LOCAL-RIN-OWNER.

This intrinsic can be used to implement a 'signaling' system whereby a process can lock a specified Local RIN to signal another process that it requires data. The signaled process could call LOCRINOWNER to find out the PIN of the requester and pass the needed data to it.

Releasing Local RINs

FREELOCRIN is used to release *all* previously acquired Local RINs. It can be called by *any* process in the tree even if that process did not originally call the GETLOCRIN intrinsic. If any of the RINs to be released are locked at the time the call is made then *none* of the RINs will be released. In this situation another call to FREELOCRIN is needed. For example:

```
CALL INTRINSIC "FREELOCRIN".
IF C-C NOT = 0 THEN
     .
     .
     .
```

This intrinsic must also be called if you need increase the number of Local RINs allocated. It makes coding your programs difficult if you intentionally allocate fewer RINs than you need; you should try to avoid this situation. On the other hand, do not allocate more RINs than you will need since this depletes the number of RINs available, especially the number available for use by MPE to lock files!

MPE automatically calls FREELOCRIN upon return to the Command Interpreter (that is, when the C.I.'s son process terminates), regardless of which process called GETLOCRIN.

FILE RINS

In addition to Global and Local RINs there are File RINs. They are taken from the same RIN Table as Local RINs, but unlike Local RINs, they are accessible from different jobs/sessions. They are acquired and used only by the File System for purposes of locking and unlocking files. If you are familiar with OPT (Online Performance Tool) you may, on occasion, have seen processes on 'GRIN' (Global RIN) waits in the

Process Context Screen. These processes were probably waiting to lock a file. Of course they could be calling LOCKGLORIN as well!

RIN CONFIGURATION CONSIDERATIONS

The number of RINs in general and the number set aside as Global RINs must be configured on your system. Changing the number of RINs will require a RELOAD and should be carefully planned.

For additional information on RIN Configuration please refer to the *System Operation and Resource Management Reference Manual*, Chapter 7, Miscellaneous Configuration Changes.

After answering 'Yes' to the MISC CONFIGURATION CHANGES? prompt you will be asked:

```
LIST GLOBAL RINS?
```

Answering 'Yes' will produce a list of the current Global RINs and the users who acquired them using the :GETRIN command. For example:

```
RIN #  USERNAME.ACCTNAME
 3     SUSAN    .DATAMGMT
 4     SUSAN    .DATAMGMT
11     OPERATOR.SYS
12     OPERATOR.SYS
```

You are then asked,

```
DELETE GLOBAL RINS?
```

Answering 'Yes' will allow you to delete Global RINs without the need to be the user who originally acquired them. This option should be used with *extreme* caution if applications on your system make extensive use of Global RINs.

The following two questions request the total number of RINs and the subset of that number allocated as Global, accessible by user application programs.

```
# OF RINS = <yy> (MIN=xx, MAX=1024, USED=xx)?
# OF GLOBAL RINS = <yy> (MIN=xx, MAX=1024, USED=xx)?
```

The configured number of RINs is used to determine the size of the 'primary' RIN Table. This table is composed of a 3 word entry for each configured RIN plus one additional 3 word entry for header information. This means that the size of this portion of the table is calculated as follows:

Primary RIN Table size = (Number of RINS + 1) * 3

The configured number of Global RINs is used to determine the size of the secondary portion of the RIN Table. This area is used to store the RIN's password and the user and account names of the user who executed the :GETRIN to acquire the RIN. Each entry is 24 bytes in length (12 words). You can calculate the size of the secondary table as follows:

Secondary RIN Table size = Number of GLOBAL RINS * 12

The total size of the RIN Table is the sum of the primary and secondary sizes. Note that File RINs only use entries from the primary RIN Table; they do *not* use any space from the secondary table.

NOTE

The RIN Table is stored on the system disc because Global RIN assignments must 'survive' system starts. This is the reason why changes to this table can only be made on a RELOAD. A fully configured RIN Table with 1024 total rins all assigned to be Global will use approximately 121 sectors on Logical Device 1. The RIN Table *is not* spread to other drives.

MULTIPLE RIN CAPABILITY

This capability is granted to programs using the MPE :PREPARE (or :PREP) command or the SEGMENTER PREPARE command. For example:

```
:PREP $OLDPASS,myprog;CAP=IA,MR
```

The only way a user can grant MR to a program is if the user possesses this capability. Additionally, a program must reside in a group that has at least the *same* capability list assigned to it as the program. *Users* of these programs *do not* require any of the special capabilities.

What Does MR Do?

Multiple RIN capability gives a program the ability to do the following:

- Lock more than 1 file concurrently.
- Lock more than 1 database or path within one database concurrently.
- Lock more than 1 Global RIN concurrently.
- Any combination of these abilities. For example: Lock one Global RIN and a database at the same time.

WARNING

If MR is not used carefully it can cause application programs to hang due to locking conflicts. It may be necessary to WARMSTART the system to clear such hangs.

For additional information on this subject please refer to the *MPE V System Intrinsic Reference Manual*, Section III, Resource Management.

SPECIAL CONSIDERATIONS

As stated earlier, the RIN Table is a general RIN 'pool' from which *all* RINs are selected. Because of this, care should be used when determining the number of Global RINs, if any, to be used, as well as the maximum number of Local RINs any process may request. The reason is that the MPE File System needs to use RINs for file locking and if there are none available you will find that programs begin aborting with GLOBAL RIN UNAVAILABLE (FSERR 60). (Note that this message is *not* referring to Global RINs! It is referring to File RINs.)

One way to prevent this is by limiting the number of Global RINs allowed. This will ensure that permanently allocated RINs do not use up all available space. Since there is no way to limit Local RIN allocation using configuration settings the best solution is to limit Local RIN allocation to a maximum value and, by convention, never request more than that agreed maximum number.

Conditional vs. Unconditional Locking

The difference between these two locking strategies is simple; in an 'unconditional' mode the process requesting the lock will stop, or IMPEDE, if the lock request cannot be granted immediately. This indicates that some other process 'owns' the RIN. In the 'conditional' mode the process does not impede; it keeps running, the intrinsic returns a 'CCG' or C-C > 0 condition code to signal *why* the request could not be satisfied. The program must differentiate between CCG, which might not be an error, and CCL, which most certainly is an error.

Why Use Conditional Locking?

In general, unconditional locking is the easiest way to request resource locks of any kind. But there are reasons for using conditional mode.

Since processes impede on unconditional locks it is possible to use an initial conditional lock attempt to find out if a resource is available. If it is not, your program could perform another function until the resource becomes available. For example, in an interactive application you could ask the user of the program to select another option or else wait until the needed resource is available.

Conditional locking can also be used to limit the use of a program to a single user at a given time. Since MPE makes no such restrictions, this limitation must be built into the program. Using a single Global RIN your program could, as soon as it begins execution, attempt a conditional lock on this RIN. If it fails, the program informs the user and ends; otherwise, the program keeps the lock for the duration of its life to prevent others from running it.

Additional Reading

The following manuals provide additional information on the topics discussed in this Note:

- *MPE/V Commands Reference Manual,* (Part No. 32033-90006)
- *MPE/V Intrinsics Reference Manual,* (Part No. 32033-90007)
- *MPE/V System Operation and Resource Management Reference Manual,* (Part No. 32033-90005)
- *MPE File System Reference Manual,* (Part No. 30000-90236)
- *KSAM Reference Manual,* (Part No. 30000-90079)
- *COBOLII/3000 Reference Manual,* (Part No. 32233-90001)
- *IMAGE Data Base Reference Manual,* (Part No. 32215-90003)
- *TurboIMAGE Reference Manual,* (Part No. 32215-90050)

Training

The following Hewlett-Packard classes are available if additional training is required in any of the areas discussed:

- A Programmer's Introduction (22801D)
- System Management (22802E)
- Advanced System Management (35017A)
- HP 3000 Special Capabilities (22805B)
- Designing and Optimizing Applications (22808B)

BACK ISSUE INFORMATION

Following is a list of the Application Notes published to date. If you would like to order single copies of back issues please use the *Reader Comment Sheet* attached and indicate the number(s) of the note(s) you need.

Note #	*Published*	*Topic*
1	2/21/85	*Printer Configuration Guide (superseded by note #4)*
2	10/15/85	*Terminal types for HP 3000 HPIB Computers (superseded by note #13)*
3	4/01/86	*Plotter Configuration Guide*
4	4/15/86	*Printer Configuration Guide - Revised*
5	5/01/86	*MPE System Logfile Record Formats*
6	5/15/86	*Stack Operation*
7	6/01/86	*COBOL II/3000 Programs: Tracing Illegal Data*
8	6/15/86	*KSAM Topics: COBOL's Index I/O; File Data Integrity*
9	7/01/86	*Port Failures, Terminal Hangs, TERMDSM*
10	7/15/86	*Serial Printers - Configuration, Cabling, Muxes*
11	8/01/86	*System Configuration or System Table Related Errors*
12	8/15/86	*Pascal/3000 - Using Dynamic Variables*
13	9/01/86	*Terminal Types for HP 3000 HPIB Computers - Revised*
14	9/15/86	*Laser Printers - A Software and Hardware Overview*
15	10/01/86	*FORTRAN Language Considerations - A Guide to Common Problems*
16	10/15/86	*IMAGE: Updating to TurboIMAGE & Improving Data Base Loads*
17	11/01/86	*Optimizing VPLUS Utilization*
18	11/15/86	*The Case of the Suspect Track for 792X Disc Drives*
19	12/01/86	*Stack Overflows: Causes & Cures for COBOL II Programs*
20	1/01/87	*Output Spooling*
21	1/15/87	*COBOLII and MPE Intrinsics*
22	2/15/87	*Asynchronous Modems*
23	3/01/87	*VFC Files*
24	3/15/87	*Private Volumes*
25	4/01/87	*TurboIMAGE: Transaction Logging*
26	4/15/87	*HP 2680A, 2688A Error Trailers*
27	5/01/87	*HPTrend: An Installation and Problem Solving Guide*
28	5/15/87	*The Startup State Configurator*
29	6/01/87	*A Programmer's Guide to VPLUS/3000*
30	6/15/87	*Disc Cache*
31	7/01/87	*Calling the CREATEPROCESS Intrinsic*
32	7/15/87	*Configuring Terminal Buffers*
33	8/15/87	*Printer Configuration Guide*