Table of Contents

Basic QIP

1.1 General Description.

1.1.1 Purpose.

QIP, a performance enhancement package for Pr1me computers, implements read-only demand paging directly from named files. This methodology is commonly used on other virtual memory computing systems, and yields several advantages over conventional PRIMOS paging:

(1) Program start-up is much quicker, since demand paging is used instead of SEG, which must read all of the program and data into memory.

(2) Sharing of procedure and read-only data occurs automatically; only when a data segment is modified is a private user copy made.

(3) As a result of these two factors, system paging activity is reduced.

(4) Less allocated paging space is required.

(5) Better balance among disk drives and controllers can be achieved.

(6) The administrative complexity and service disruption entailed by using "shared segments" under PRIMOS is eliminated.

The effectiveness of QIP over SEG generally increases in proportion to the size of procedures and the number of concurrent users.

1.1.2 Compatibility.

The QIPLOAD utility converts existing SEG runfiles into a paging format (QIP format). Ordinarily, no changes to the source program or to the SEG load procedure are required by QIP; the package can be installed and used with minimum effort and training. Optimization techniques are available to secure even better QIP performance.

1.2 Installing QIP.

1.2.1 Initial installation.

Using MAGRST, restore the QIP directory from the supplied magnetic tape, then type "ATTACH *>QIP.UFD" and then type "R INSTALL".

1.2.2 Configuration changes.

Before QIP will run, a directive must be added to the CONFIG file used for PRIMOS coldstarts, and the system must then be coldstarted. The directive is:

```
NVMFS 400      /* 256 VMFA segments.
```

It must be placed prior to the "GO" directive in the CONFIG file. The NVMFS directive is a special PRIMOS directive which will not have any adverse effect on system integrity or performance.

PLEASE NOTE: If there is an NSEG directive in CONFIG, the sum of the parameters to NSEG and NVMFS must be less than '1776 (octal). If there is no NSEG directive, the adjustment will occur automatically.

1.2.3 Initialization.

In specific instances, pertinent only to PRIMOS revision 19 or above, a special initialization program must be run from the operator console to permit QIP to operate. The two instances when this is required are:

(1) On any system operating under PRIMOS revisions 19.0 or 19.1.

(2) If the QIP run-time option, -SHARE, is to be used.

The initialization command must be invoked under the control of the operator terminal, preferably by adding to the coldstart procedure file (PRIMOS.COMI):

```
R SYSTEM>QIPINIT
```

QIPINIT has no effect if it has been previously run, or if it is run under PRIMOS revision 18. QIP software delivered at PRIMOS revision 18 does not include QIPINIT.

1.3 Converting from SEG to QIP: QIPLOAD.

To convert a SEG runfile into a QIP runfile, specify:

```
QIPLOAD {SEG-runfile-pathname} {QIP-runfile-pathname}
```

If both command arguments are omitted, QIPLOAD will prompt for them. If the QIP runfile is omitted, its name will default from the SEG filename.

QIPLOAD obeys the PRIMOS filename suffix conventions. For example:

```
QIPLOAD RUN>EXAMPLE.SEG
QIPLOAD RUN>EXAMPLE
QIPLOAD RUN>EXAMPLE.SEG   RUN>EXAMPLE.QIP
QIPLOAD RUN>EXAMPLE.SEG   RUN>EXAMPLE
QIPLOAD RUN>EXAMPLE       RUN>EXAMPLE.QIP
QIPLOAD RUN>EXAMPLE       RUN>EXAMPLE
```

all will convert "RUN>EXAMPLE.SEG" into a QIPfile named "RUN>EXAMPLE.QIP".

QIPLOAD never opens the SEG runfile for writing.

1.4 Running a QIP program.

1.4.1 Normal invocation.

Resuming a QIP program is very similar to resuming a SEG program. Either

```
QIP  RUN>GIGANTIC
QIP  RUN>GIGANTIC.QIP
```

will result in the execution of "RUN>GIGANTIC.QIP", since QIP, like QIPLOAD, recognizes suffix conventions.

Generally, start-up of a QIP program requires noticeably less time and fewer page faults than the SEG copy of the identical program. Paging is effected directly from the QIP file, and multiple users of the same file will share both paging space and high-speed memory.

In the event that a procedure or data segment is modified, QIP will automatically and invisibly revert the segment to the traditional paging mechanism, avoiding interference among sharing users. Ordinarily, all procedure segments and perhaps some data segments will utilize QIP paging during the entire program execution.

1.4.2 Error recovery.

QIP restores the normal PRIMOS environment when the application program returns to PRIMOS for any reason (including most errors). Nevertheless, specifying

```
QIP
```

without any command arguments will insure that environment is completely restored following abnormal program exits, abends, etc.

Intermediate QIP

2.1 Theory of operation.

QIPLOAD. QIPLOAD is more of a converter than a loader. The SEG mechanism for creating SINGLE/SHARE files is used to create files which are in a segment directory rather than being named files, and which are DAM rather than SAM files (this is a requirement of the file system paging mechanism). There is one file for each non-empty segment originally specified in the SEG load. A descriptor file (file 0) contains pertinent information for QIP, and also retains information from the SEG load.

QIP. QIP processes the user command line optionally consisting of pathnames of QIP files, and command arguments prefixed with a hyphen. QIP establishes a relationship between specific segments and their files in the QIP segment directory which causes all read and execute references (including I/O) on those segments to page from the files in the segment directory, rather than using the conventional paging mechanism. The segments are write-protected such that if the program attempts to modify a QIP segment, an ACCESS_VIOLATION$ fault is generated, and QIP silently takes control and manifests the segment to conventional paging with all access rights available. Until that occurs, the portion(s) of the program which are paged from the QIP file will be shared among concurrently accessing users (both paging space and high-speed memory).

Because QIP establishes limited access to normally used segments (such as segment '4001), the potential for access violation faults exists even after the QIP program has exited. For this reason, QIP attempts to detect all situations which will return to PRIMOS. Ordinarily, such situations as errors will be trapped by QIP, and all of the segments will be manifested to conventional paging, such that sharing is then disabled, but the program is still executable (using the PRIMOS "START" command). Some errors, as well as FORTRAN STOP statements, will cause QIP to "shutdown" its program environment. This is quicker than manifesting all of the shared segments, but it is effectively a "DELSEG ALL" which prevents program resumption. Program exit control and cleanup is adjustable via the -CLUP command argument (see below).

QIP is designed to operate as a more efficient replacement for SEG, and is usable from keyboard, COMINPUT, or CPL just as SEG is. QIP is ordinarily silent and undetectable by the user. However, through use of command line options -VPSD, -DEBUG, and -ONUNIT additional information may be obtained from QIP to permit the system administrator or programmer to assess and tune system performance.

2.2 Optimizing QIP performance.

By design, use of QIP does not mandate that either program source changes or even changes to the SEG load in order to operate. Nevertheless, opportunities for optimizing performance under QIP do exist, none of which will impair program operation under SEG.

2.2.1 Initialized common blocks.

An "initialized common block" is a FTN/F77 common block which contains one or more variables which are initialized before binding using the DATA statement. PL/I programs accomplish the same effect by using the INIT and EXTERNAL attributes in a variable declaration. Initialized common blocks decrease compile and load speed, and require more disk space under both SEG and QIP, than uninitialized common blocks.

The following recommendations are made for situations where initialized common blocks are required:

(1) Separate (using SEG's loader's SYMBOL, A/SY, and R/SY commands) the common blocks so that any read-only common blocks are in different segments than other common blocks. Read-only common blocks (such as prompts, error texts, and constants) will be shared by QIP among all concurrent users.

(2) Specify (using SYMBOL, etc.) initialized common blocks contiguously, and place them at lower addresses in the segment than uninitialized common blocks. This technique minimizes the disk space required by SEG and QIP to store the initialization data.

2.2.2 SEG's loader's MIX command.

All Pr1me translators output separated procedure and data object text as a result of compilation. Ordinarily, procedure text is unmodified during program execution, while data text by definition is always modified. Accordingly, QIP can share procedure text among concurrent users, while data text must be "copied" from the QIP runfile and maintained per user under conventional paging. SEG'S loader's MIX command defeats the separation of procedure and data, making the program "impure" (ie., not sharable at all). Therefore, use of MIXed loads under QIP is not recommended.

2.2.3 Condition handling.

As described above, QIP intercepts all situations which invoke PRIMOS command input. At that time, QIP self-disables the direct paging and sharing mechanism, unless exit/cleanup has been modified by the user

using the -CLUP option.

If the application or PRIMOS should signal a condition, and there is no on-unit for that condition in the QIP-invoked application, QIP will intercept the condition before it reaches PRIMOS, and perform its exit/cleanup sequence. QIP will always continue signalling the condition to PRIMOS.

However, if there is an on-unit for the condition, and the on-unit is able to handle the condition (rather than continuing to signal to PRIMOS), QIP will not "see" the condition at all. For example, some applications utilize the BREAK key as a standard means of escape to a known point in a program. By writing an on-unit for "QUIT$", the condition signalled when BREAK is depressed, the programmer can prevent QIP from self-disabling as a result of this situation. Details of the condition mechanism are described in PDR 3621, PRIMOS Subroutines.

If a QIP application intends to signal a condition to be serviced by a CPL on-unit which invoked the QIP application, the condition will, in this case, be recognized by QIP. If this is not desirable, an appropriate QIP -CLUP option will have to be specified.

IMPORTANT: Read details in "Advanced QIP" before writing an on-unit for the condition "ACCESS_VIOLATION$".

2.2.4 IFTNLB.

Pr1me's "impure FORTRAN library", IFTNLB, is normally loaded into a procedure segment by SEG. Since IFTNLB programs are impure (ie., write on themselves), any segments which contain IFTNLB routines will become unshared as a result of the execution of these routines, which include subroutines for FTN built-in functions SIN, COS, LOG, etc.

To sidestep this inefficiency, IFTNLB may be specific-loaded by SEG into either a known data segment, or a previously unused segment. An excerpted example:

```
$ PL                    /* loads the pure FTN library.
$ S/IL 0 4070 4070      /* loads impure lib into unused seg.
```

2.3 Special considerations.

2.3.1 Use with DBG.

It is not possible at this time to utilize Pr1me's Source Level Debugger, DBG, with a QIP format file. Because the nature of debuggers requires that the procedure text be modified from time to time, debugging would generally negate the efficiency features of QIP anyway.

However, it is possible to QIPLOAD and QIP a program which has been compiled and loaded with any combination of routines using -DEBUG and -PRODUCTION compile modes. Because QIPLOAD does not copy the debug information, there is no difference between runfiles created under -PRODUCTION and runfiles consisting of routines with no debug options at all.

2.3.2 Stack assignment and location.

To maximize its own efficiency, QIP controls stack segment assignment, ignoring all stack-related parameters which may have been supplied at SEG load time. Because QIP has complete knowledge of the user address space, it allocates stack an entire segment at one time using the first available segment at the top of the user address space, and automatically extends the stack by one segment whenever stack overflow occurs.

This variation from SEG will never cause a problem unless the application program contains special programming which is sensitive to the location or size of the stack. Automatic overflow, which is almost always desirable, can be disabled or modified by writing an on-unit for condition "STACK_OVF$".

2.3.3 "File in use" message on QIP runfiles.

If an attempt is made to delete or write on a QIP runfile which is activated by QIP, a "file in use" message will result. This message will result even though a display of STATUS UNITS may not reveal that the file is "open" on any unit by any user. Similarly, using "CLOSE pathname" from the operator terminal will not change the "file in use" status. This is because the file is "in use" by the paging system, and will continue to be in use until all programs referencing it have "released" the association of the file with memory segments (QIP cleanup).

2.3.4 Changing the name of a QIP runfile.

Even though the QIP runfile may be in use, only the segment subfiles are actually active for the duration of execution; the segment directory itself is open only briefly at the invocation of the QIP run. Accordingly, the name of the QIP runfile (segment directory) may be changed while its subfiles are in use. This feature is quite useful for installing new or corrected versions of a program without disrupting the sessions of users in process.

2.3.5 SHUTDN (the operator command).

If the operator command SHUTDN is issued for a disk partition containing an active QIP file, the message "Segment xxxx deleted." will appear, perhaps numerous times, on the user's terminal with the appropriate segment number substituted for "xxxx". Thereafter, if the deleted segment is referenced, it will be "zeroed" and the application program is likely to err or crash as a result. Consequently, care must be taken to avoid placing QIP files on disk partitions which are likely to be shut down in the course of normal operations.

2.3.6 "OUT_OF_BOUNDS$" errors.

An unfamiliar diagnostic may appear from a QIP run which will not arise from a SEG run. This message begins:

```
Error: condition "OUT_OF_BOUNDS$" ...
```

and is otherwise similar to "ILLEGAL_SEGNO" and other usually undesirable messages. It arises only from attempts to read or execute on a currently shared segment at an address which is higher than the highest address specified for that segment in the SEG load.

Because no data has been written at this location (otherwise the segment would not currently be shared), the message is a bonafide and reliable indication of a logic error in the application program; it implies reference to an "undefined" area.

Even though such an error might not interfere with the results obtained, there is no appropriate action QIP may take (the referenced area is beyond "end-of-file" on the paging file); therefore, the application program must be fixed.

2.3.7 Application processing of a command line.

QIP allows a variety of options on its command line; a fact which could conflict with some target applications which also process elements from the command line. To handle this situation in an orderly way, QIP recognizes the command argument "-E" as the end of its own command list. If the application program specifies the argument to "get next token" for the RDTK$$ subroutine, any and all information following "-E" will be supplied. To avoid processing QIP arguments, the application must never "rewind" the command line.

2.4 QIP -DEBUG option.

QIP is able to supply status information allowing a knowledgeable observer to assess performance and environmental considerations relating to a specific QIP run. The -DEBUG command line argument requires a string of one or more digits in any order which specify what features are to be displayed. The meanings of the digits are as follows:

0. Sound the terminal's "bell" (tone) whenever a segment is copied from shared to private paging. This is useful for monitoring "copypage" operations under FORMS or other delicate situations.
1. Display "Go" immediately prior to program entry.
2. Display a message whenever the RELEASE or SHUTDOWN cleanup routines are invoked.
3. Display all encountered signals which are not ignored by QIP.
4. Display the segment/word address of references which cause a segment to be copied from shared to private paging.
5. Display the segment address of each stack segment allocated.
9. Display the KST (segment usage) map prior to execution.

For example:

```
QIP PROG -DEBUG 512
```

will activate debug options 1, 2, and 5 and might look like this:

```
Go
QIP Stackseg 4374(3)/4.
QIP shutdown.
```

If your installation has debug options configured into QIP, this may be reversed by specifying -NODEBUG on the command line, followed by the numbers of the debug options which are to be suppressed.

2.5 QIP -CLUP option.

QIP incorporates intelligence to determine which of three cleanup actions it will take if it is about to relinquish control to PRIMOS. Depending on the condition, QIP will either discontinue the user environment (this operation is called a "shutdown"), or discontinue the sharing and direct paging while preserving the execution environment (this operation is called a "release"), or ignoring the condition. Because of differences among programs and installations, some tailoring or modification of the cleanup mechanism may be dictated. The QIP -CLUP command line argument must be followed by a single character indicating the cleanup action to occur:

D specifies the default cleanup mode, which will "release" on most conditions, but will "shutdown" on detecting program EXIT, STOP, or any condition which is not returnable. This option is used to override a different cleanup option which may be configured in an installation's version of QIP.

R specifies that a "release" is to be done at any time either "release" or "shutdown" would normally occur. This option maximizes the restartability of the program from PRIMOS, although such restarts and reenters will be to a program which is not shared, ie., the restarted program will operate entirely under conventional paging.

S specifies that a "shutdown" is to occur any time QIP encounters a condition or program exit. This option executes much quicker than "release", although it precludes restartability from the PRIMOS command level. The assumption behind this option is that users will always abandon the run if any "fatal" error is encountered.

N specifies that "release" and "shutdown" will NOT occur under any circumstances. This option is the "fastest" of all, and it is useful if BREAKS, arithmetic exceptions, and other conditions are expected regularly and if restartability from these "errors" is a requirement. It is also necessary if conditions are signalled from an application to CPL, and "release" is not desired. This option retains restricted access to normally unrestricted user segments, even after the QIP program is "done". Accordingly, anomalous behavior will result if this option is used, and if a non-QIP application is subsequently run without an intervening "QIP", "DELSEG ALL", or "LOGOUT"/"LOGIN" following the QIP run.

2.6 QIP default BREAK key handling.

If BREAK (or control-P or equivalent) is depressed during a QIP execution, and if the program does not have an on-unit which completely handles the "QUIT$" condition, QIP will display the following:

```
   QUIT. QIP program suspended at 4444(3)/55555. Continue?
```

If the user types "YES", "Y", or "OK" the program will continue as if nothing happened. Otherwise, QIP will perform the appropriate cleanup routine (see -CLUP, above) and will continue to signal the condition (presumably to a CPL or to PRIMOS).

This QUIT query feature is intended to allow pacing or program monitoring without necessarily sacrificing the performance advantage of QIP. It also recognizes that a BREAK will occasionally arise which was not intended by the user, due to a keying error or line transmission noise.

Nonetheless, a user or installation can suppress the default QUIT alert message, by specifying -NOQUIT on the QIP command line. On the other hand, if -NOQUIT has been configured into your installation's QIP, you can revert to reporting QUIT conditions by specifying -QUIT on the QIP command line.

2.7 QIPLIB: interprogram serial linkage.

There are four QIP library subroutines which facilitate invocation of a QIP runfile by a QIP runfile. This capability is useful for assembling relatively independent program modules, perhaps using a central control or menu program. Linking is chain-style; the invoked program does not "return" to the invoking program. This feature allows all of the programs in a chain to utilize the entire address space, without particular attention to address overlap unless a common mailbox is to be shared between the programs. Ordinarily, a simple SEG load can be used for each QIP file to be linked, an advantage over the more powerful multiple QIP file capability (described in "Advanced QIP").

To be able to use the QIP library routines, the SEG load procedure for the program(s) involved must specify "LIB QIPLIB" or its equivalent. These subroutines are usable only within the QIP environment. If the SEG runfile is used, these calls will generate an error message.

QIPLNK effects a transfer of control from on QIP runfile to another. The QIPLNK calling sequence is intended for PL/I usage:

```
Declare QIPLNK Entry
    (fixed binary(15), character(*) varying, fixed binary(15))
Call QIPLNK (MODE, TEXT, CODE)
```

* MODE defines the linkage mode; in this version MODE must be zero.

* TEXT may contain any information acceptable to the QIP command (normally only the name of the QIP runfile to invoke).

* CODE is a standard PRIMOS error code returned only if the arguments are invalid. QIP errors (such as file not found) will utilize the normal QIP error-handling facilities.

QIPLNF effects a transfer of control from one QIP runfile to another. The QIPLNF calling sequence is intended for FORTRAN and COBOL usage:

```
INTEGER MODE, TEXT(4), LENTXT, CODE       /* FTN example */
DATA MODE /0/, TEXT/'NEXTFILE'/, LENTXT/8/
CALL QIPLNF (MODE, TEXT, LENTXT, CODE)
```

The arguments MODE, TEXT, and CODE have the same meaning as for QIPLNK; argument LENTXT is the character length of argument TEXT. A COBOL example follows:

```
77 MODE    PICTURE S9999, USAGE IS COMPUTATIONAL, VALUE IS ZERO.
77 TEXT    PICTURE X(8),  USAGE IS DISPLAY, VALUE IS 'NEXTFILE'.
77 LENTXT  PICTURE S9999, USAGE IS COMPUTATIONAL, VALUE IS 8.
77 RTNCODE PICTURE S9999, USAGE IS COMPUTATIONAL.

CALL 'QIPLNF' USING MODE, TEXT, LENTXT, RTNCODE....
```

The linkage subroutines, QIPLNK and QIPLNF, have no provision for argument passing among the linked runfiles. A "mailbox" common block could be established at a specific address known to all of the programs, so that the runfiles could communicate with each other. However, QIP cleans the entire address space between each QIP runfile invocation including linked invocations, thus destroying any residual data. This mechanism can be selectively suppressed, facilitating program to program communication.

To establish a mailbox, the program language must be able to define external symbols for data (ie., FORTRAN common, PL/I external, etc.) which precludes COBOL usage at this time. The external structure must be defined identically in each cooperating program and must be given the same name. Finally, the SEG loader's SYMBOL command must be identically specified in the SEG load for each cooperating program, for example:

```
    SYMBOL MAILBX 4020 1000
```

Generally, the SYMBOL command should precede all LOAD and LIB commands.

To inform QIP that the area is to be saved, the address (or pointer) of the area, and its length in characters, must be supplied to the QIP monitor. To do this, use QIPSAV prior to calling QIPLNK or QIPLNF. FTN and PL/I examples follow:

```
      COMMON /MAILBX/ ITEM1, ITEM2, REAL1, REAL2, DOUBLE, ETC
 C    Character counts:  2      2      4      4       8    4   =  24
      INTEGER*4 MBSIZE
      DATA MBSIZE /24/

      CALL QIPSAV (LOC(MAILBX), MBSIZE)

      ------------------------------------------------------------

      Declare QIPSAV Entry (Pointer, fixed binary(31))
      Declare MAILBOX Character(20) static external
                                    init ('QIPSAV TEST.  123456')

      Call QIPSAV (Addr(MAILBOX), 20)
```

Of course, the designer may elect to use other methods of inter-program communication such as semaphores, disk files, or shared memory, none of which are interfered with by the QIP system.

Advanced QIP

3.1 Multiple QIP files: interprogram concurrent linkage

It is possible to have several QIP runfiles active simultaneously; for example, a menu program named X may need to invoke "main" programs ALPHA, BETA, DELTA, and GAMMA, each of which must be able to be invoked independently by name. Other QIP menu runfiles might also invoke any of ALPHA, BETA, DELTA, or GAMMA. Ordinarily, under SEG, these four programs would be loaded as subroutines into each and every runfile in which they might be invoked. Although QIP relieves the previous problem of long start-up time for such applications, a tiny change to GAMMA, for instance, results in a long and tedious reload procedure for every module which references the changed routine.

To cause several QIP runfiles to be simultaneously resident in memory, the several QIP runfile names are all supplied on the QIP command line:

```
QIP X ALPHA BETA DELTA GAMMA
```

Possibly some, or all, of these routines will actually be executed; the "main" routine, X, always will be started by QIP. Program X then must decide which, if any, of the other routines will be executed. Up to sixteen (16) QIP runfile names may be specified on the command line, and any QIP options must follow the runfile names.

Address space planning must be exercised to be sure than there are no segment overlaps among the procedure or linkage of runfiles which are to be concurrently specified to QIP (similar to the '2xxx public segments). Uninitialized common blocks may be specified in more than one of the SEG loads; attention to alignment (using the SYMBOL command) may be required. To effect inter-program linkage, each QIP runfile must be supplied with the address of the ECB or ECBs which it can reference. This is done with the SYMBOL command.

An example on the next page demonstrates how two cooperating processes are loaded using SEG, and executed using QIP.

Example of two cooperating QIP runfiles.

```
OK, SEG -LOAD     /* First load F2, the secondary program...
[SEG rev 18.3]
$ S/LO F2 0 4021 4022       /* Two segments "reserved" for F2.
$ D/LI                      /* CAREFUL! Must be a ditto load.
LOAD COMPLETE
$ MAP
*START  4022  000002  *STACK  7777  000000  *SYM    000023

SEG. #    TYPE        LOW      HIGH      TOP
  4021    PROC      001000    001214    001214
  4022    DATA      000000    000053    000053

ROUTINE        ECB         PROCEDURE     ST. SIZE  LINK FR.
  F2       4022  000002   4021  001000    000042  000026  4022  177400

DIRECT ENTRY LINKS
  EXIT     4021  001172    TNOU     4021  001176    TNOUA    4021 001202
  F$WA     4021  001206    F$CB     4021  001212

OTHER SYMBOLS
  F183KWIK  4021  001143

$ QUIT               /* We now know F2 ECB is at 4022/2.
OK, SEG -LOAD        /* Now load the primary routine, F1...
[SEG rev 18.3]
$ SYMBOL FRIEND 4022 2   /* The subroutine name for F2 is FRIEND.
$ S/LO F1 0 4011 4012    /* Two different segments reserved for F1.
$ D/LI
LOAD COMPLETE
$ QUIT                   /* The hardest part is now done!
OK, QIPLOAD (F1 F2)      /* MAKE TWO QIP RUNFILES.
F1.QIP load complete.
F2.QIP load complete.
OK, QIP F1 F2

This is the first part of two segmented loads which
are designed to test inter-linking capabilities in
the QIP software.  Following is output from F2...

This output is from program F2.  Program F2
will return, which will either exit or return
to the invoker.

This is program F1 again... SIGNING OFF.

OK,
```

3.2 QIP -SHARE option: interprogram public linkage.

An advantage of QIP is the ability to achieve sharing and to minimize program start-up overhead without resorting to explicit management of the public segments (octal numbers less than '4000). Nevertheless, there may be situations where the use of public address space may still be required for reasons of addressing compatibilities, user created shared libraries, or permanent residency in virtual memory. QIP accomodates the use of public segments.

QIPLOAD will transfer all segments from the SEG runfile to the QIP runfile. However, QIP will not attempt to associate the public segments with their QIP segment files, since operations on public segments are privileged. Instead, QIP will only associate private segments with their segment files, assuming that the public segments have been previously initialized. This assumption is consistent with the present PRIMOS mode of operation.

However, if the -SHARE option is supplied on the QIP command line, only public segments will be associated, and the QIP program(s) specified will not be started. This feature is analogous to the PRIMOS SHARE command and, similarly, it is usable only from the operator terminal.

For example:

```
    QIP PROGS>DEMO -SHARE     /* executed from the operator terminal.

        ... then subsequently ...

    QIP PROGS>DEMO            /* operable from any terminal.
```

It is unlikely, but possible, that -SHARE is configured as a permanent option in your version of QIP. If this is the case, normal QIP execution can be restored by specifying -NOSHARE on the QIP command line.

IMPORTANT: Access privileges to QIP public segments must not be changed using the SHARE command! If WRITE access is defined for a QIP segment at PRIMOS revision 18, and if a write access to the segment should occur, the QIP runfile will be altered, and possibly damaged.

3.3 QIP -ALLOCATE option.

QIP reserves segments in the user space for QIP operation during program execution. These segments are typically assigned from the highest addressable segment downward, but in a way that will not conflict with procedure and data segments used by the QIP program(s). Segment conflict resolution is not normally applied to "empty" uninitialized segments specified in the SEG load, but containing no information. (Such segments appear on the SEG map with LOW address 177777 and HIGH address 000000.) Segment conflict resolution can be applied to these uninitialized segments as well, by specifying -ALLOCATE (or simply -A) on the QIP command line. This option imposes a start-up time penalty which is proportional to the number of uninitialized segments in the runfile(s).

NOTES:

QIP allocates all of its own segments prior to starting the QIP runfile. QIP never needs additional segments once the program is underway.

Some programs indiscriminantly use "unused" segments, which are not referenced in the SEG load map. Such usage is not detectable or predictable by QIP, and -ALLOCATE will not help. The programs involved will have to be modified to use documented segments only, or to test segment usage. The PL/I ALLOCATE and FREE statements exhibited such indiscriminant usage at PRIMOS revision 18.3; use of QIPLIB in the SEG load of applicable PL/I programs will load improved versions of the applicable routines (which are also usable under SEG).

If the -ALLOCATE option is configured to be on in your QIP, it may be disabled by specifying -NOALLOCATE or -NOA on the QIP command line.

3.4 QIP -VPSD option.

This feature is analogous to the SEG {program} 1/1 feature of SEG. It invokes a VPSD built into QIP prior to transfer of control to the QIP runfile, but after QIP's internal tables and on-units have been initialized. If VPSD is used to modify data in the shared QIP segments, QIP will "COPY_SEG" the segments into private address space just as if the QIP program itself had made the modification.

If -VPSD is configured into your QIP, you may disable it by specifying -NOVPSD on the QIP command line.

3.5 Condition handling in the QIP environment.

QIP relies extensively on the condition mechanism to provide proper transition from the restricted QIP addressing environment to the normal unrestricted environment. The presence of this condition handling software is ordinarily invisible to both users of QIP programs and to the system administrator, with two exceptions:

(1) Performance aspects discussed in "Intermediate QIP", above;
(2) User handling of "ACCESS_VIOLATION$".

3.5.1 General method.

QIP establishes an on-unit for ALL conditions which reach it via the PRIMOS condition mechanism. This on-unit "decides" which cleanup method to use, unless this has been specified on the user command line. The following condition names are ignored at or before PRIMOS revision 19.1, because they always either return or signal other conditions which will be "seen" by QIP or the application:

```
ARITH$
COMI_EOF$
SETRC$
PH_LOGO$
NULL_POINTER$
LISTENER_ORDER$
SUBSYS_ERR$
ENDPAGE
FINISH
NAME
STRINGSIZE
```

QIP provides special services for the condition "STACK_OVF$" (stack overflow), by allocating stack space in known, available segments. QIP also specially services "QUIT$" to provide BREAK key handling appropriate to the QIP environment. Finally, "ACCESS_VIOLATION$" is specially handled to discern write attempts on QIP segments, invoking the "COPY_SEG" operation when this occurs; otherwise, the on-unit simply continues the signal to PRIMOS.

3.5.2 QIP -ONUNIT option.

By specifying -ONUNIT on the QIP command line, somewhat more control can be exercised by a knowledgeable user running a QIP program interactively. If -ONUNIT is active, all on-units signalled, except for those listed above as "ignored", will generate a message from within QIP itself and provide a list of actions:

```
Options:
[C]ontinue program,
[P]RIMOS,
[H]elp (this text),
[M]ap of QIP segs.,
[D]isplay the stack,
[V]PSD.
```

The first two options, C and P, will either continue the program from the point of interruption (if possible) or continue the signal to PRIMOS, respectively. The other options provide the appropriate information or facilities, and then they redisplay the option list.

3.5.3 User written on-units.

When QIP encounters a signal, it is not possible to determine or control whether the user, at PRIMOS command level, will resume the execution environment using the PRIMOS START command, or will simply execute another program. Accordingly, QIP will tidy the memory environment and disable QIP sharing unless instructed otherwise using the -CLUP command line option.

If specific signals are expected during the routine usage of a QIP program, it may be desirable to provide pre-programmed on-units to handle those signals. Also, user-written on-units developed in the SEG environment should operate equally well in the QIP environment. In fact, on-units which completely handle a condition (and do not continue the signal) will prevent the condition from even being "seen" by QIP.

Except for the condition "ACCESS_VIOLATION$", there are no known restrictions governing the use of the condition mechanism under QIP.

If -ONUNIT is configured into your copy of QIP, the feature may be reversed by specifying -NOONUNIT (or simply -NOON) on the QIP command line.

3.5.4 User handling of "ACCESS_VIOLATION$" conditions.

User handling of "ACCESS_VIOLATION$" conditions is not recommended under the QIP environment. If an on-unit for this condition is truly necessary, the on-unit must continue the signal if the fault address in the fault frame header is a DTAR 2 address (ie., segments in the range from '4000 to '5777). Any messages normally displayed under these circumstances should be suppressed, since the "ACCESS_VIOLATION$" condition is very routine in the QIP environment, and is handled silently by QIP.

QIP Configuration

Nearly all of the QIP command line options can be permanently installed in QIP itself (ie., QIP.SAVE on CMDNC0) through use of the -CONFIGURE (abbreviation: -CONFIG) option. This option will save all option settings which are simultaneously supplied on the command line with it, so that frequently used options need not be specified on the QIP command line every time QIP is used.

The action of the configuration mechanism is cumulative, ie., if one configuration operation follows another, and the second one does not cancel any options set by the first, then all of the options declared will be set. To undo the result of a configuration option, all of the configurable options except for -CLUP may be prefixed by "NO" (-CLUP is simply set to one of its four valid settings).

Here is a table of valid option settings, abbreviations, and negations:

O P T I O N	N E G A T I O N
-DEBUG, -D	-NODEBUG, -NOD
-CLUP, -C	<none>
-SHARE, -S	-NOSHARE, -NOS
-ALLOCATE, -A	-NOALLOCATE, -NOA
-VPSD, -V	-NOVPSD, -NOV
-ONUNIT, -ON	-NOONUNIT, -NOON
-QUIT, -Q	-NOQUIT, -NOQ

NOTES:

(1) The specification of -DEBUG or -NODEBUG will only affect the debug options corresponding to digits following -DEBUG or -NODEBUG.

(2) The user may, at run time, activate or negate any option which has been configured into QIP.

(3) Three QIP command line arguments:
-CONFIG[ure]
-H[elp]
-E[nd]
are not configurable options.

(4) The user of the -CONFIGURE option must have write access privileges to the QIP save file. Security conscious installations may wish to establish read-only access to this file to prevent unauthorized configuration.

SUPPLEMENTAL INFORMATION - QIP

January 25, 1984

This document supplements the QIP Instruction Guide. It explains features and corrections available in QIP systems shipped on or after January 25, 1984. These features and corrections are:

1. QIPINIT (reference 1.2.3). QIPINIT is only required on systems running PRIMOS revisions 19.0 and 19.1, or on PRIMOS revision 19.2 and up when the QIP -SHARE option is used. QIPINIT need not be run on systems using PRIMOS revision 19.2 or higher, if the -SHARE feature is not used.

2. QIP environment. QIP formerly performed a "DELSEG ALL" when invoked. This operation zeroed memory segments not used by QIP, but which could potentially hold user defined inter-program information. Effective with this release, QIP only deletes all segments when invoked without arguments (reference 1.4.2). When QIP is invoked with one or more file names, a segment is claimed for each file name and starting at the top of user memory address space (as defined with the CONFIG NUSEG directive). Then, each non-empty segment loaded by SEG is claimed, and a corresponding segment number is claimed, resuming from the top of user memory address space. For an example, simply invoke any QIP program with -DEBUG 9, and note the segment assignments presented in the ensuing KST map.

To avoid the possibility of over-writes of inter-program communication areas, these areas should be assigned to the lowest numbered segments possible. Better yet, CONFIG NUSEG should be set sufficiently large to insure that the QIP temporary segments (assigned in top-down sequence) do not encroach on any areas expected to be defined by application programs.

3. QIP linkage (reference 2.7). Multiple linkages aborted (with differing errors, depending on the PRIMOS revision) from P$ALC. The linkage mechanism was recoded to avoid using the PL/1 allocate mechanism, and is now capable of an indefinite number of linkages.

4. QIP configuration sensitivity error. At PRIMOS revision 19, if the CONFIG directive NUSEG was set to less than the allowable maximum (normally, octal 360), QIP would issue either a "BAD PARAMETER" diagnostic, or abort with an "ILLEGAL_SEGNO$" error. This "memory-top" sensing mechanism is now fixed.

5. QIP, new option (reference 2.6). The default BREAK key message issued by QIP can be controlled using two new command options to QIP:

-QUIT, -Q	to enable quit servicing by QIP (default), and
-NOQUIT, -NOQ	to inhibit quit servicing by QIP.

This feature is configurable (reference section 4).

6. QIPMON. Some garbled message texts were corrected.