# Harlequin RIP

## The Harlequin Extended Error Handler

Technical Note Hqn017

October 2006

GLOBAL GRAPHICS®

# 1 Introduction

Most versions of the Harlequin RIP have access to a text-based standard output channel, and a large proportion of copies of the RIP are used for high resolution output where extended error reporting on the output medium is not a desireable option. Harlequin have therefore released an extended error handler which shows all its results by default on the standard output channel. The information reported is sufficient to allow a PostScript programmer to isolate most problem code as rapidly as possible, and there are some configuration switches to change the amount of information reported.

It is assumed that the reader will be familiar with PostScript language programming, and with the normal error handling procedure in a PostScript interpreter (as described on pages 100-101 of the PostScript Language Reference Manual (second edition)).

The error handler is shipped with RIP version 3.2 revision 27 and later, and with RIP version 3.3 revision 6 and later.

# 2 Using the error handler

In most cases the most efficient method of using the error handler is by installing it in a page feature. Two sample page features which show long and short format reports are included as **`SW/Page Features/Examples/Error Handler - Long and ~Short (SW\PAGEFEAT\EXAMPLES\ERRLONG`** and **`ERRSHORT`** on FAT-based machines). Obviously these cannot be used as they stand to debug problems with other page features; to do that similar code could be added to a file run at RIP start-up, or into the top of the problem page feature.

# 3 What's in the procset?

The procset itself contains four procedures:

dict **`LoadErrorHandler -`**

Installs the error handler and configures it according to values supplied in the dictionary. Configuration options are described below.

**`– QuickErrorHandler -`**

Installs the error handler with default options for short-form reports.

– `FullErrorHandler -`

Installs the error handler with default options for long-form reports

– `ClearErrorHandler -`

Removes the error handler if it is already installed.

# 4 Configuration

Current configuration options are:

`longErrorReport`

A boolean which should be `true` if a complete stack report is required or `false` if only the top few items on the operand stack should be displayed and reporting of dictionary, execution and file stacks should be suppressed.

`fullDictReport`

An integer which defines the largest dictionary which will be fully reported if its name cannot be determined. See the description of the operand stack in the error handler report below for more information.

`HqnShowOperand`
`HqnShowDictionary`
`HqnShowExecution`
`HqnShowFiles`

Set these to `true` to show or `false` to disable the individual stack information reports.

Default: `true`

`HqnShowFollowing`

If positive, show the following lines section in the file stack with this number of bytes.. Set to 0 to suppress that section.

Default: `512`

`HqnShowStrings`

Set the maximum length of strings (in bytes) to show in the operand stack. Any string over this length will be truncated and displayed with '`...`' after it. Use `-1` for no limit.

Default: `128`

`HqnShowSubStrings`

Set the maximum length of strings (in bytes) to show in for strings inside arrays or in the dictionary stack list

Default: `64`

Set how various classes of characters should be shown in strings:

`HqnNormalChars`

Normal ASCII' (<20>-<FE>)

Default: `raw`

`HqnPrintControls`

'Useful' control bytes (BT, CR, LF)

Default: `raw`

`HqnControlChars`

Other low-byte characters (<00>-<1F> and <7F>, excluding HT, CR and LF)

Default: `allhex`

`HqnHighByte` High-byte characters (<80>-<FF>) in non-UTF-8-compliant strings.

Default: `raw`

`HqnUTF8` High-byte characters (<80>-<FF>) in UTF-8-compliant strings.

Default: `hex`

Values of each should be:

| | |
|---|---|
| `raw` | Just show the character for that byte in the current font. |
| `dot` | Replace with a dot ('.') |
| `oct` | Report in octal with a leading '\' character |
| `hex` | Report as a block of two hex digits, surrounded by '<' and '>' |
| `allhex` | The whole string should be shown as a hex string if any relevant bytes occur. |

In addition, the following may be used *only* for HqnUTF8

| | |
|---|---|
| `unihex` | Convert multi-byte UTF-8 sequences to a unicode code point, and report that as a block of 4 hex digits, surrounded by '<' and '>'. |

Additional otions are:

`HqnResolveJDF` If the string provided matches the host-server file name of a file submitted through the JDF Enabler, it will be replaced by the URL supplied in the JDF, prefixed with "J:" when this is `true`.

Default: `true`

`HqnResolveURLPercent`

If the string contains any 3-byte sequences matching the percent escape used in URLs (for example, "`%20`" for a space) they will be resolved to the referenced byte value before display, and the string will be prefixed with "P:" if this is true.

Default: `false`

**HqnShowArrays**

Set the maximum number of bytes to show of Arrays in the operand stack. Any array (executable or literal) over this length will be truncated and displayed with '`...`' after it. Use `-1` for "no limit"

Default: `768`

**HqnShowDictArrays**

Set the maximum number of bytes to show of Arrays in the dictionary stack list and in dicttionaries in the operand list:

Default: `256`

**HqnShowExecTop**

**HqnShowExecBase**

The top and bottom of the execution stack are usually suppressed because they do not provide valuable information. Occasionally, the determination of where the relevant portion of the stack is can be incorrect. These switches allow the whole execution stack to be shown.

Default: `false`

These options must be passed to a call to **`LoadErrorHandler`** in the preceding dictionary.

In addition the **`recordstacks`** entry in the **`$error`** dictionary will be acted on.

If the **`print`** operator has been re-defined then output may be redirected to other text channels, or even printed on the output medium if required. Such re-definitions must be done using `shadowop` in order to act on the bound instances of **`print`** in the error handling routines.

# 5 The error handler report

An example long-format report is shown below. As you can see it is divided into a header, and then reports on the operand, dictionary, execution and file stacks. The stack information will not be reported if **`recordstacks`** in `$error` is false, or if the error was **`VMerror`**. The entire error handler report will be suppressed if the error was `interrupt`.

This example is from a PostScript file which was deliberately changed to trigger an error.

```
****************** ERROR HANDLER REPORT (Long format)
******************
ERROR:              /undefined
OFFENDING COMMAND: wibble
CONFIG INFO:        []
CURRENTGLOBAL:      false
VMSTATUS:           3 405764 2826432
CTM:                [8.33333 0.0 0.0 -8.33333 -0.0 6600.0 ]
CURRENT FONT:        (W-R+L)    "Courier"  DLD1  [10.0 0.0 0.0 -
10.0 0.0 0.0 ]

OPERAND STACK: 4
****0:(realtype) 27.6667
****1:(booleantype) false
****2:(integertype) 0
****3:(dicttype)  (W+R+L)  userdict(length 71 / max 256)

DICTIONARY STACK: 6
****0: (W+R+L)  statusdict(length 146 / max 200)
****1: (W+R+L)  UNKNOWN DICT  -  <<
    *** xpqA - 0.166667
    *** xpct2 - ()
    *** xprh - {xpfg 6 0 --put-- xpih --aload-- --pop-- --setdash-
- }
    *** setpageparams - {letter xppop2 0 --exch-- 792 --sub-- --
neg-- --translate-- --pop-- }
    *** xppopcolor - {xpcustmprsnt {xppop2 } {xpcmykprsnt {xppop4
} {--pop-- } --ifelse-- }
--ifelse-- }
    *** ...    -     ...
    >>  -  (length 202 / max 300)
****2: (W+R+L)  md(length 223 / max 270)
****3: (W+R+L)  userdict(length 71 / max 256)
****4: (W+R+G)  globaldict(length 2 / max 128)
****5: (W-R+G)  systemdict(length 378 / max 512)

EXECUTION STACK: 4
****0:{1080 1584 0 1 setpageparams {} settransfer --end-- pxs pys
--scale-- ppr --aload-- --pop-- por {--pop-- --exch-- --neg-- --
exch-- --translate-- --pop-- } {--translate-- --pop-- --pop-- 270
--rotate-- } --ifelse-- 1 -1 --scale-- }
****1:{{--newpath-- --clippath-- --mark-- {--transform-- {--
itransform-- --moveto-- } } {--transform-- {--itransform-- --
lineto-- } } {6 -2 --roll
-- --transform-- 6 -2 --roll-- --transform-- 6 -2 --roll-- --
transform--
 {--itransform-- 6 2 --roll-- --itransform-- 6 2 --roll-- --
```

```
itransform--
 6 2 --roll-- --curveto-- } } {{--closepath-- } } --pathforall-- -
-
newpath-- --counttomark-- --array-- --astore-- /gc xdf --pop-- }
--exec
-- }
****2:file: "%EP12Quadra950%ErrorJob.PS"
****3:--stopped--

FILE STACK:
****:"%EP12Quadra950%ErrorJob.PS" - realfile, open, line: 507,
byte: 24642
  **FOLLOWING LINES:
%%EndDocumentSetup
%%Page: ? 1
op
0 0 xl
1 1 pen
0 0 gm
(nc 0 0 1584 1080 6 rc)kp
0.64314 0 0.24705 setrgbcolor
1 1 xppen
 0 xps
Job Not Completed: HX3fx; document: MT class page
```

***The header*** contains a few details about the error itself and the current state of the interpreter.

The configuration info will almost always be simply `[]`.

Currentglobal, vmstatus and CTM are as returned by the `currentglobal`, `vmstatus` and `currentmatrix` operators.

The current font is reported by name, font type, access and its `ScaleMatrix` entry, which may be used to determine its size, and rotation and any shearing etc. The access information is displayed in a standard format which will be used throughout the report:

The first element describes whether the dictionary is write protected or not, W- means it is write-protected, W+ means that it can be written to.

The second element describes whether the dictionary is readable or not, using R+ and R- in the same way as for write-protection.

The third element describes whether the dictionary is in local or global VM – L means local and G means global.

Thus the `(W-R+L)` access in the example above means that the font dictionary is write protected, but may be read and is in local VM.

***The operand stack*** describes the elements which were on the operand stack when the operator which caused the error was called. The `****` is present simply to allow the start of each operand to be found easily when some of them are rather long, as they can be if they are arrays or PostScript language procedures. The numbers immediately after the asterisks record the position in the operand stack – 0 is the top of the stack.

The value and type of each object is then reported , together with additional details depending on the type.

The access (write protection, read protection and globalness) is reported for all compound objects (arrays, procedures, strings, dictionaries) as described above for the current font.

Where possible the lengths of arrays, procedures and strings are reported. The lengths and maximum lengths of dictionaries are shown. If any compound object is read-protected then these are omitted.

Wherever possible the names of dictionaries are determined as the job is interpreted, but in some cases the dictionary is not named at all (e.g. the job contained something of the form of **`6 dict begin`**), or the name of the dictionary cannot be intercepted. In these cases the dictionary is reported as an `UNKNOWN DICT` and the first few items of the dictionary are listed to allow some attempt at identification to be made. The items are obtained by a call to the PostScript language `forall` operator, which means that their selection may vary from time to time when the job is run. The maximum number of entries to be reported is defined by **`fullDictReport`** as described above. The default value for long format reports is six, meaning that any dictionary containing six or fewer values will be reported in full. If the dictionary contains more than six items then five will be reported, followed by `"... - ..."`. The format for items within the unknown dictionary simply shows the name and value of each item. An example of how an unknown dictionary is displayed is shown above in the dictionary stack part of the report.

***The dictionary stack*** shows the dictionaries which were open at the time the offending command was called. The dictionary reported first was the top on the stack. The format of these reports follows that described above for the operand stack except that the type field is omitted because all entries are assumed to be dictionaries.

***The execution stack*** shows those operators, files, procedures and parts of procedures which were queued for execution when the error occurred. The stack is automatically truncated so as not to show parts of the full stack which are within the error handler itself or in the server loop. Those parts will not be relevant to determining the cause of a problem in a job or page feature. The truncation used will always ensure that the final entry is the `stopped` operator from the server loop.

***The file stack*** shows all the files in the execution stack in more detail. The type of each file is reported, together with whether it is open or closed and the line and byte within the file to which it had been read when the error occurred. File type may be:

realfile – opened with the **`file`** operator on an ordinary device.

editfile – either **`%lineedit%`** or **`%statementedit%`**. This will only ever been seen while running the executive.

filterfile – opened with the **`filter`** operator.

stdfile – the file is open on one of the pseudo-devices **`%stdin%, %stdout%`** or **`%stderr%`**.

The next 128 characters after the point to which the interpreter had read the file are then displayed if the file is open and not an editfile.

An equivalent short format report would resemble the following.

```
****************** ERROR HANDLER REPORT (Short format)
******************
ERROR:             /undefined
OFFENDING COMMAND: wibble

OPERAND STACK: 4
****0:(realtype) 27.6667
****1:(booleantype) false
****2:(integertype) 0

FILE STACK:
****:"%EP12Quadra950%ErrorJob.PS" - realfile, open, line: 507,
byte: 24668
  **FOLLOWING LINES:
%%EndDocumentSetup
%%Page: ? 1
op
0 0 xl
1 1 pen
0 0 gm
(nc 0 0 1584 1080 6 rc)kp
0.64314 0 0.24705 setrgbcolor
1 1 xppen
 0 xps
Job Not Completed: HX3fx; document: MT class page
```

In the default short format report the header is reduced to the error and the offending command. Only the first three items on the operand stack are shown, and the dictionary and execution stack are completely suppressed. Only the first file in the file stack is reported.

# 6 Interpreting the report

The first step in using the error report will usually be to determine the point in the file that the interpreter had reached when the error occurred. Both the line and the byte reported in the file stack are those to which the interpreter had read the file when the error occurred; it is entirely possible, indeed highly probable, that the cause of the error is not at that point in the file, but finding which procedure is called at that point may well be of assistance in further investigations.

The line number reported takes note of all line end formats which are valid in a PostScript language file (line feed, carriage return or carriage return/line feed), not including those within binary data. If a file contains mixed line ends

or images in binary format then line numbers in a text editor may not coincide with the numbers reported here. It is often easiest to find the correct place in the file by searching for the code which is reported as immediately following the error. In the example file we can find only one such section:

```
T T 0 0 1584 1080 0 0 1584 1080 100 72 72 1 F F F F T T T F psu
(HX3fx; document: MT class page)jn
0 mf
od
%%EndDocumentSetup
%%Page: ? 1
op
0 0 xl
1 1 pen
0 0 gm
(nc 0 0 1584 1080 6 rc)kp
0.64314 0 0.24705 setrgbcolor
1 1 xppen
 0 xpsg
64 xpgr
```

The error report shows that the next token to be read from the file is `op` on a line of its own. Thus the error will probably have occurred during the execution of `od`, just before the comments. If any comment parsing is in effect then it's possible that parsing the **`%%EndDocumentSetup`** or **`%%Page:`** comments could have triggered the error.

Whilst most programmers will use their own techniques, and different techniques may prove most appropriate for different errors, a productive next step is often to search for code matching that reported as the top item in the execution stack. In this instance we can find:

```
/xpappendod{bind /xpappndary 4 array def xpappndary 2 3 -1 roll
put
  xpappndary 0 [/od load /exec load] putinterval xpappndary 3
[/exec load] putinterval
  /od xpappndary cvx store}def
{statusdict begin
userdict 0 false 27.66666 wibble
  1080 1584 0 1 setpageparams {}settransfer end
 pxs pys scale ppr aload pop por
 {pop exch neg exch translate pop}
 {translate pop pop 270 rotate}ifelse 1 -1 scale
}xpappendod
```

The offending command in the error handler report was wibble, and the top entry in the execution stack report matches the code immediately following the `wibble` in this piece of code. In addition the code immediately preceding it will leave items on the operand stack which match the report. Finally it appears that the code in question is being added to the end of an existing procedure named `od`, which ties back to the position to which the interpreter had read the file. It would appear that we have found the code which triggered the error.

This example is a very simple one, where at least the final cause of the problem could be found rapidly and easily. In many cases considerably more detective work is required – even here we would have to ask why **`wibble`** was undefined and yet used in the code.

Other techniques such as trace emitted to the System Monitor by debugging code added to the job which reports the value of specific variables, or uses `pstack` to show the current state of the stack may obviously be used to supplement the information obtained from the error handler.

Once the problem has been found it may also be necessary to correct it, but that is not an area within the scope of this document.

| Change history | | |
|---|---|---|
| v 1.0 | 94.11.21 | Created |
| v 1.1 | 2001.06.12 | Updated cover page and copyright page.<br>Removed references to ScriptWorks and replaced with Harlequin RIP.<br>No other changes made to text. |
| v1.2 | 2006.10.04 | Minor corrections. Copyright updated. |