Harlequin RIP

Developing Color Plugins for the Harlequin RIP

Technical Note Hqn042

June 2001

1 Introduction

This technical note is a supplement to the *Harlequin RIP Plugin Kit: Guide for OEMs* (and to a lesser extent, *Using Harlequin RIP Extensions*), describing how OEMs can create or modify output plugins to work most efficiently with RIP version 5.0 or later.

Note: This information is intended for OEMs only. This issue (v1.2) adds a note on Euclidean screening for PhotoInk devices (in Section 8.3 on page 28) to v1.1's minor corrections and other clarifications relative to a draft issued as v1.0. All changes are marked by vertical bars in the left margin, as shown here. Please report any problems that you find in using this document.

We recommend that you read as much as possible of this document, and relevant parts of the other guides referenced, before planning your work. This document contains both general hints and some specific information not yet documented in the appropriate the Harlequin RIP manuals. We strongly suggest that you read all of Section 2 through Section 7 on page 21.

Conventions: This document uses terms in square brackets, such as [RB3] and [SWPLUGIN], to refer to external documents; all such terms and the full references are listed in Appendix C, "" on page 57.

1.1 Contents

The remainder of this document contains the following major sections:

- A summary of reasons for enabling plugins to use the features of the Harlequin RIP. See Section 2.
- Some applications of N-color plugins and a review of what an output plugin contains and does. See Section 3 on page 4.
- Modernizing a plugin that uses a previously supported color space. See Section 4 on page 7.
- Extending a modern plugin to support a HiFi N-color space. See Section 5 on page 13. Section 6 on page 20 then discusses some application-specific differences in jobs that support N-color.
- A comparison between plugins supporting HiFi and photo-ink color spaces. See Section 7 on page 21.

- Section 8 on page 25 through Section 12 on page 33 describe subsidiary topics such as screening, Roam, limiting excessive ink delivery, and color management.
- Appendices containing material that will become part of the Harlequin RIP Plugin Kit manual and Harlequin RIP Extensions manuals. See Section A on page 35 and Section B on page 38 for details.

2 Motivation

The primary reason for making a plugin require the Harlequin RIP is that it then has access to features of PostScript LanguageLevel 3. The best known feature is probably the DeviceN colorspace but there are other benefits.

- Raster formats can describe additional color spaces for output by the plugin. Collectively, [RB3] section 4.8.4 calls the additional color spaces DeviceN. Harlequin calls each specific device color space a *colorant family*. The **`rasterFormat`** structures and their accompanying **`ColorantInfo`** structures describe colorant families to the RIP using the **`D_GET_RASTER_FORMAT`** and **`D_GET_COLORANTS`** selectors.
- The Page Setup (and Edit Style) dialog box is now the preferred way to select interleaving style, raster depth, and separation style, by choosing from the raster formats offered by the plugin. This means that the plugin can be simpler because its Configure Device dialog box no longer has to support choices of this sort.
- Frame-interleaved and band-interleaved modes can now have an arbitrary number of colorants instead of only three (RGB, CMY) or four (CMYK). For example, it is possible to construct two-color band interleaving or a seven-color frame interleaving with six process colors and a spot color. (Note: At Harlequin RIP version 5.1, pixel-interleaved mode still supports only three or four process colors and only at 8-bit depth per colorant.)
- Photo-ink support: the plugin can declare that though the colorant family supports multiple colorants, some are different densities of the same hue. This simplifies operations such as calibration and color management, which can still be done in CMYK for the current implementation or, in principle, some other well-known colorant family. (Note:

Only supported at RIP version 5.1 and later. RIP version 5.1r1 and earlier do not support EasyTrap for photo-ink colorant families, but this limitation is expected to be temporary.)

- Color devices can now support the output of separations. This is automatic and works for unmodified plugins. Separations include colored separations where the separation is drawn in the color it would be printed in conventionally and progressives where more than one colored separation is present on a page, each page adding a color in turn.
- New operators allow dynamic control over separations and the colorants to be produced.
- Another, and less obvious, benefit is that separations may now separate out not only colors, but also certain object types selectively. For example, one can request only cyan components of images to be present on a separation.

There are other changes, described later where they affect the process of implementing a plugin, but the above summarize the benefits to the user and developer.

Many existing plugins using CMYK or specialized output color spaces can work correctly with RIP version 5.0, but they may be operating inefficiently or losing the full benefits of operating with RIP version 5.0, because they are using compatibility operators within the Harlequin RIP.

Why change?

- Existing plugins may work in an inefficient way because facilities included with the RIP were not previously available.

  For example, a number of plugins for devices using specialized output color spaces use portions of the Harlequin RIP imposition package to achieve the required data formats and are then unable to use the other features of the imposition package for things such as filling the output media efficiently or drawing cropmarks. A rewrite of such a plugin to

use the recommended facilities for Harlequin RIP version 5.0, will provide the same features while working more efficiently and freeing the imposition package for its intended use.

Roam would not have displayed such rasters usefully. Using modern interleaving styles can cure this deficiency in most cases.

- It is now possible to support more colorants for new systems such as six-color printing: for example, Hexachrome and photo-ink.
- It is possible to simplify the writing and maintenance of plugins by using facilities which were not directly available before.

3 New and old ideas

The ultimate reference for the PostScript language is now [RB3]. The most relevant sections for the color space applications and handling discussed in this document are: Section 4.8, “Color spaces”; and Sections 7.1 through 7.3 on rendering, conversion, and transfer functions.

There are some applications of N-color within the RIP that are not obvious from [RB3]. The following subsections list these applications, suggest how to implement N-color plugins, and give some background on the relationship between the RIP and an output plugin in terms of files and data flow.

3.1 N-Color plugins

There are many varieties of N-color, some dealing with additional visible colors and some with control channels. Here is a summary of the color output formats that the Harlequin RIP can now support with appropriate plugins:

- Conventional: CMYK, RGB, Gray.
- Photo-ink: CMYKcm, CMYKcmk.
- HiFi: Hex, CMYKOG, CMYKRGB.
- Special purpose: masked channels, special or custom colors.
- Spot colors in single raster: CMYK + Gold, CMYK + varnish.
- Two-color forms printing.

Note: The notation CMYKcm illustrates a common convention for photo-ink systems — to use uppercase and lowercase letters to distinguish between normal and light versions of the same colorant.

This document concentrates on the HiFi and Photo-ink cases but these two cases serve as useful examples for the others.

3.2 How do I get N color channels?

Harlequin suggests that you approach this in two stages, at least for existing plugins that support an established color space (CMYK, RGB, or Gray). The two stages are:

- Modernize the plugin
- Extend to N-color

Doing the work in these two stages gives you some ability to test your work at a stage when a modernized plugin should perform similarly to an older plugin working with Harlequin RIP 4.x. If your modernized plugin shows problems or produces different output before you add N-color output, you can be reasonably confident that you have done something wrong while working with the fundamental links between the Harlequin RIP and the plugin, or in handling a well-known color space.

Before listing the steps that we suggest that you should follow in more detail, it may be helpful to review the links between an output plugin and the RIP. There are at least two ways to look at these links:

- Directories and files
- Data flow

3.3 Directories and files

The complete package for a color-managed plugin consists of:

- The plugin executable itself
- Aliases for colorants
- A calibration target and profile
- A setup file of PostScript-language code

- Miscellaneous files used by the plugin

[SWPLUGIN], Section 2.11, describes a folder structure which can be used to group all of these files together.

3.4 Data flow

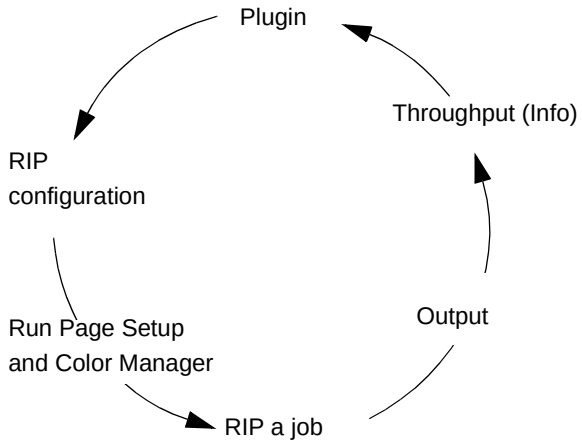


Figure 0.1 How the RIP and plugin interact

Figure 0.1 shows a plugin communicating its capabilities to the RIP, using **`rasterFormat`**, and the availability of these options to the user in the dialog boxes where a user can create devices and create page setups that use the new devices. Subsequently, when the RIP processes a job it generates a page buffer that holds the appropriate headers including device parameters, and data in the appropriate format.

If the RIP is running in a multiple mode, the page buffer passes to the throughput system and is visible to the user in the Output Controller / Monitor dialog box. (The user can find out some details of the page buffer in the Info dialog box and the subsidiary Configure Device dialog box. Many plugins allow the user to change some details of the page buffer or its rendering in these dialog boxes, but this ability can be mildly dangerous in some circumstances. For example, selecting the **Negative** check box can give a non-colorimetric result.)

When the page buffer is output to the plugin, the plugin has to interpret the data given to it and carry out the transport and conversion duties necessary to feed the final output whether that is a communication channel to a physical output device or a file.

4 Modernizing a plugin

Even if you have no existing plugin to consider, follow these steps to see how to create a plugin to support an established output color space (CMYK, RGB, or Gray) using the recommended methods for the Harlequin RIP version 5.x. The subheadings follow roughly the stages in the loop shown for data flow between the plugin and RIP, shown in Figure 0.1, page 6.

Note: A plugin can be written to support use with both RIP version 4.5 and version 5.x, but it would require very careful handling of plugin interface versions.

4.1 Inform the RIP

The plugin informs the RIP of its color capabilities by setting up appropriate values in the **`rasterFormat`**:

- Use **`colorType_general`** for **`colorType`**.

  For example, a plugin might have previously used the value **`colorType_CMYK_frame_interleaved`**. The information about the packing format is now given in separate fields with the **`rasterFormat`** when using **`colorType_general`**.

- Fill in the new **`rasterFormat`** entries for **`interleavingStyle`**, **`rasterDepth`**, and so on. (See the header file **`gdevdefs.h`**.)

- Set **`numFixedColorants`** and **`numAliases`** to zero.

  This indicates to the RIP that the color space to be used is an established (PostScript Level 2) one and means that the RIP will not call the new **`D_GET_COLORANTS`** selector, which is only needed for more complex colorant families (or to allow reordering or addition of colorants), and will use **`colorValues`** to obtain information about the order of the colorants.

- **pColorantFamily** must be one of: **DeviceGray**, **DeviceRGB**, or **DeviceCMYK**

  This means that the colorant names (Cyan, Magenta, and so on) are implied and need not be declared explicitly, as you will later see is *required* for N-color.

  Note: The *contents* of the **pColorantFamily** string must be copied, from the plugin to the location pointed at by the RIP. Do not copy a *pointer* and thus overwrite the pointer to refer to a string within the plugin.

- Set **pCalibrationcolorantFamily** to an empty string, *not* a null pointer.
  prf -> pCalibrationcolorantFamily [0]=0;
- Set **pCustomConversions** to an empty string, *not* a null pointer.
  prf -> pCustomConversions [0]=0;
- Set **calibrationAsPress** to FALSE. This is required for all versions of the RIP version 5.1 and later. (See Section A.1 on page 35 for documentation of this field, which was called **calibrationIsSeparating** in the Harlequin RIP 5.1r0.)

Remove the equivalent setup (for Harlequin RIP 3.x and 4.x) from the plugin PostScript code. Generally, this means that you *must* remove all references to **Tones**. Having a definition referencing **Tones** will overwrite **rasterFormat** definitions made in a modern (Harlequin RIP 5.x) manner by triggering compatibility behavior, as described in [SWEXTN], Section 8.11.

Note: A plugin that uses **colorType_general** and that allows the RIP to offer color style selection instead of providing its own options in the Configure Device dialog box no longer needs to support the **D_WHICH_RASTER_FORMAT** selector.

4.2 Make the plugin directory-structured

[SWPLUGIN], Section 2.11, describes the use of a directory or folder within the **SW/Devices** folder to hold all files associated with a plugin. This is required for DeviceN output color spaces and Harlequin strongly recommends that you follow this practice in implementing all your plugins.

Why do this? There are several reasons:

- All the plugin information is in one place, and readily identified with the plugin. For example, there is no danger of overwriting a support file if the user subsequently installs another plugin that happens to use a support file of the same name.
- It provides a well defined location for profiles, targets, screening information, and so on.
- The **`%fs%`** i/o device provides PostScript-language code with a shorthand method of referring to files within the folder. For example, you can write **`%fs%Setup`** in place of **`SW/`***`plugin`***`/Setups/Setup`**. In some cases **`%fs%`** also implements a search path among device-specific, plugin-specific, and system-wide default files. See [SWPLUGIN] for details.
- It enables localization of plugin dialog boxes and messages to support different languages, when running with a localized version of the RIP.
- It enables future developments because of the ease of adding new subfolders and setup and localization files. (One example of such a potential development is the possibility of defining Java-based dialog boxes.)

4.3 Separate different color formats into different device types

Having separate device types for each supported color format will make it easier to organize appropriate calibration sets, calibration targets, and profiles so that the user sees a small set of relevant choices for each device.

There is a possible complication when you add device types. If the plugin now becomes a multiple device plugin providing several device types, the RIP no longer instantiates (creates) a device for each device type. It becomes necessary to instantiate a device for the device types. The user can do this in the Device Manager dialog box but you may prefer to have the plugin request the RIP to automatically instantiate some or all devices. See [SWPLUGIN], section 3.6 for details.

Consider using automatic instantiation if there is a small set of device instances that all your users are likely to use regularly. This reduces the setup effort required by the user and produces devices with known names and capabilities, which may simplify subsequent configuration, training, and other support.

If there is not an obvious set of frequently-used devices, do not auto-instantiate all devices.

4.4 Change the plugin to receive new raster formats

The data arriving from the RIP is described differently when **`colorType_general`** is used and may have some new formats. The plugin must analyze the **`pageHeader`** to work out the quantity and significance of the arriving data. The tasks for the plugin include:

- Cope with **`colorType_general`**:

  The plugin should no longer infer anything about the raster format from the **`colorType`**, just check that it is **`colorType_general`** and look to other fields for details.

- Use the new **`pageHeader`** fields for **`interleavingStyle`**, **`rasterDepth`**, and so on, instead of inferring these details from the value of **`colorType`**, as in earlier versions of the RIP.

- Perhaps, use the **`pColorants`** linked list, especially to handle unmapped channels, as explained next.

If the plugin previously used the **`colorValues`** field to determine the order of the colorants, it can now find this information represented in the **`pColorants`** linked list hanging off the **`pageHeader`** structure.

`pColorants` describes the colorants present in the raster. These may be different from those published by the plugin because:

- The RIP may be separating. Separation is appropriate to all devices in the RIP version 5.x.

- The user or the job may have added additional spot colors or, if the plugin allowed it, some colors may have been removed because no marks were made in that color.

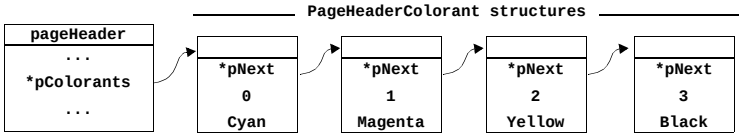
Each stream of data in the plugin — frame of frame-interleaved mode, band of data for a single colorant in band-interleaved mode, or single color in monochrome — is known as a *channel*. Colorants are said to be *mapped* to channels. Not all channels need have a colorant mapped to them and, when they do, it need not be the color you expect.

An example may help you understand these possibilities.

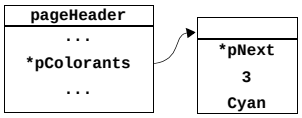
Consider an inkjet printer typically driven in CMYK band-interleaved mode. This has four channels. That the first channel, numbered 0 (zero), produces cyan is a property of the hardware. Data for all the channels will normally be present, unless the plugin has said that channels can be omitted. If channels cannot be omitted, they may sometimes contain only blank data.

The colorants in **`pColorants`** give a channel number to which they are mapped.

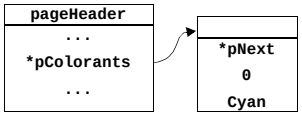
So, for an ordinary composite print, one would see each color mapped to its natural channel:



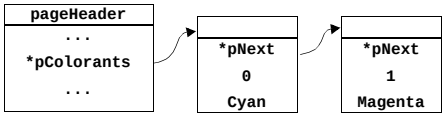
If the user asked for separations, the plugin sees only a single colorant, mapped to the black channel:



If the user asked for colored separations, the cyan would map instead to the first (cyan) channel:



If the user asked for progressives, the plugin sees the same first raster as for colored separations but the second raster is likely to be:



If the plugin does nothing other than deliver its four channels to the relevant nozzles of the inkjet we are considering in this example then it does not need to look at these linked lists, because unmapped channels still present data (which is blank).

Nevertheless, these lists are available and the plugin should examine their contents when the colorant must be known: for example, if the plugin names an output file according to colorant or if the plugin needs to ask for the colorant to be loaded on the printer, perhaps as a donor sheet handled by an operator.

Note: The necessary screens are the same as those required in RIP version 3.x or 4.x.

4.5 Test that the plugin does what you expect

This is a good point to check that the plugin behaves correctly:

- In the Separations Manager, check that the expected default color styles exist for your plugin and that you can add any extra styles that you wish the user to be able to create.

  Note: The RIP names default color styles in the simplest unambiguous way: for example, if the plugin supports only frame-interleaved styles, the RIP does not include the word **`Frame`** in the name of every available style. The user can create new styles of any name.

- Check that the Edit Style dialog box contains the colorant names that you expect.

  Also, if you have written the **`rasterFormat`** exactly as suggested in Section 4.1 on page 7, check that the dialog box does not allow you to reorder or add to the list of colorants.

- Check that the Edit Style dialog box offers the expected screen names for halftone devices. Check that there are no screening options for contone devices; if there are, check the **`screensRequired`** field in the **`rasterFormat`**.
- Check that you can calibrate the device. This should include the ability to print process and spot colors together or separately on calibration targets and to select the colors to measure with Genlin and then **Import** into the Edit Calibration dialog box. See Section B.3 on page 47 for an outline description of the procedure.
- If you have modernized an existing plugin, compare the output from a range of real jobs produced by the old and new plugins. Use jobs that specify colors in all reasonable ways, including named spot colors and device-independent methods.

If the plugin passes all these tests, you have a good basis for adding DeviceN support such as HiFi or photo-ink devices.

5 Extension to HiFi color

By HiFi color, we mean the general case of N-color with independent colorants. Some parts of this procedure are relevant to photo-ink systems, but it is easier to see what is necessary with independent colorants before seeing how photo-ink systems provide a simplified system to the user.

5.1 Inform the RIP of the plugin's N-color capabilities

In the **`rasterFormat`**:

- Use **`colorType_general`** for **`colorType`**.

  This is a general rule for the Harlequin RIP 5.x.

- Set **`numFixedColorants`** to N, the number of output colorants.

  This means that there is a subsequent call to the **`D_GET_COLORANTS`** selector, when the plugin must return information about its colorants in **`ColorantInfo`** structures.

- Set **numAliases** to 1 or more.

  Each colorant can be described by one or more alternative names or *aliases*, for reasons explained in Section 5.1.1 on page 14. **numAliases** says how many alternatives the plugin provides for each colorant.

- Set **pColorantFamily** to an appropriate name.

  At Harlequin RIP version 5.1, Harlequin plugins use **Hex** for a CMYKOG family, and **PhotoInk** for a CMYK-based photo-ink system. Harlequin has prepared corresponding screens ready for use.

- Set **pCustomConversions**

  The string should reference an appropriate set of PostScript-language procedures to act as conversion functions from DeviceGray, DeviceRGB, and DeviceCMYK to the DeviceN color space. See Section 5.4 on page 16 for details.

- Set the contents of **pCalibrationColorantFamily**

  This item defines the colorant family for the color space in which the RIP will perform calibration. The value should be the same string as **pColorantFamily** (except for photo-ink devices, described in Section 7 on page 21).

5.1.1 Why would I want aliases?

Aliases are alternative names for colorants. There are many possible naming schemes and several are already in use. It is not clear that any one scheme will (or should) win universal usage.

The first alias is important in the RIP because it is the name shown in the Edit Style dialog box, and it identifies the screens to use for halftone output.

The RIP uses the following names as first aliases in the Hex colorant family:

Hex Cyan
Hex Magenta
Hex Yellow
Hex Black
Hex Orange
Hex Green

You *must not* assume that Hex Cyan is the same ink as Cyan in a DeviceCMYK system. In this case, the introduction of orange and green inks as new independent colorants can mean changes to the other colorants to obtain the best possible gamut. Furthermore, the name Cyan is reserved within the PostScript language to refer only to the Cyan of DeviceCMYK and it would cause confusion if Cyan was also used as a colorant of a DeviceN space. For example, many applications use the presence of Cyan to indicate that the colorant family is DeviceCMYK and such applications would fail.

Supply as many aliases as you require. Typically, the RIP adds two shorter aliases, for example Hex Cyan has the aliases HexC and HC.

Note: Hexachrome is a registered trademark of Pantone Inc., and rights to use this trademark are reserved.

How are colorants related to other plugins? Once a plugin, using its **`rasterFormat`**, has declared a colorant family to the RIP, that family is known to all other plugins installed in the RIP. The other plugins are unlikely to support composite output to that family but they will be able to produce separations.

Harlequin expects, at some future date, to provide a colorant editor that will enable a user to add new aliases and colorant families. However, at all revisions up to and including Harlequin RIP version 5.1r1, only a **`rasterFormat`** supplied by a plugin can introduce new aliases and colorant families.

The plugin can also use spot colors, with their names introduced by any standard PostScript language mechanism.

Hint: Because these elements are stored, a RIP in which you have run a plugin with a faulty definition is permanently corrupted. The only sure way of returning to a known state is to install a new RIP and then to install a corrected plugin.

5.2 Implement **`D_GET_COLORANTS`** selector

This provides the RIP with the names and aliases for the colorants of the color family, as described in [SWPLUGIN], section 5.12. It also provides descriptions of the colorants and whether or not they can be omitted. The underlying method of transferring this information is a number of **`ColorantInfo`** structures, described in [SWPLUGIN], section 4.8.

5.3 Consume the raster

Remember that the frame-interleaved and band-interleaved data has the appropriate number of frames or bands, not just three or four. The plugin must expect this appropriate number of channels of data, which may be a number other than the value of N for the colorant family. This "appropriate number" is the same in both sentences but may be more or less than N, because of omitted process colors or added spot colors.

Note: N-color pixel-interleaved data is not supported. Where the plugin must deliver pixel-interleaved output in N-color, use band-interleaved delivery to the plugin and have the plugin code select appropriate data.

5.4 Custom conversions

In creating the **`rasterFormat`**, described in Section 5.1 on page 13, you had to supply custom conversions. What are these?

These conversions are similar to PostScript language tint transforms, except that they convert from conventional input color spaces to N-color output spaces such as Hex (or some variant of photo-ink, described later).

[RB3] says, in section 7.2.5:

> Conversions from DeviceRGB, DeviceCMYK, or DeviceGray to a DeviceN space are performed by device-dependent means, which generally are not under PostScript program control.

Custom conversions provide these device-dependent means for the RIP.

`pCustomConversions` is a PostScript-language string which when executed leaves an executable array of three procedures on the stack. In this array:

- Element 0 (zero) is the procedure handling DeviceGray to DeviceN conversion. This procedure must consume from the stack a single value between zero and one, where zero is black as is normal in DeviceGray. It must return as many values as there are colorants in the DeviceN space, with each value being in the range 0 through 1, where 0 indicates n colorant and 1 indicates maximum colorant. (The ordering for the returned DeviceN space is defined by the **`ColorantInfo`** structures obtained from **`D_GET_COLORANTS`**. For **`Hex`**: it is CMYKOG, G topmost.)

- Element 1 does the same job for DeviceRGB to DeviceN. It consumes three values for red, green, and blue, with blue being topmost on the stack.
- Element 2 does the same job for DeviceCMYK to DeviceN. It consumes four tint values for cyan, magenta, yellow, and black, with black being topmost on the stack.

Warning: Only 64 characters are allowed in the **`pCustomConversions`** string. The most likely implementation is to use this allowance to run a file, probably best held in the **`Misc`** folder within the plugin folder, and accessed as **`SW/Devices/`***Plugin***`/Misc/`***filename*. (The use of **`%fs%`**PluginMisc to identify the correct location is not possible because the code is run outside the context of the plugin.)

The procedures have two functions:

- There is a required function of consuming the correct number of provided values from the stack and producing the correct number of output values on the stack. For example, a conversion from CMYK to Hex (CMYKOG) could leave the CMYK values on the stack and add two zeroes, as values for Orange and Green. The final result is legal (with the six values being CMYK zero zero), but does not make the most of the Hex output space.
- The other, desirable but not essential, function is to redistribute colors from the input space to the output space to make best use of the gamut of the output device.

For example, in the case of six-color output, the first procedure must take one value and produce six. Similarly, the second procedure must take three values and produce six, and the third must take four values and produce six.

Note: The Gray and RGB spaces are additive, while CMYK values are tints. CMYKOG values are also tints so you should subtract the Gray and RGB values from one before doing any other manipulation.

For example, if HexBlack is colorant number three, counting from zero, a very simple gray conversion could be:

```
{ 0 0 0 4 -1 roll 1 exch sub 0 0}
```

A typical place to place this PostScript code is **SW/Devices/***Plugin***/Misc/***file-name* and the Harlequin convention is to name the file the same as the colorant family.

Here is the full custom conversion to the Hex colorant family for the example plugin, opeg3, supplied in the Plugin Kit. As stated in the comments, there is no attempt to redistribute colorants:

```
%!PS-Adobe 3.0
% File executed to build custom conversion procedures for Hex
% Each procedure must return 6 values on the stack which represent
% the input color in the new colorspace. These conversions
% are not accurate but do satisfy the input/output requirements.

% That is:
% DeviceGray       k -> c m y k o g
% DeviceRGB    r g b -> c m y k o g
% DeviceCMYK c m y k -> c m y k o g
%

{
  {0. 0. 0. 4 -1 roll 1 exch sub 0. 0. }       % DeviceGray
  {3 {1 exch sub 3 1 roll} repeat 0. 0. 0.}    % DeviceRGB
  {0. 0.}                                     % DeviceCMYK
}
cvlit

%eof
```

This example performs a more complex transformation, but only for CMYK to CMYKOG. It distributes some Cyan and Yellow to HexGreen, and some Magenta and Yellow to HexOrange:

```
% Custom conversion routines for Hex printing.
% Produce on the stack:
%   hexcyan hexmagenta hexyellow hexblack hexorange hexgreen

5 dict begin

/CtoHG 0.2 def
/MtoHO 0.2 def
/YtoHO 0.2 def
/YtoHG 0.2 def
```

```
{
  % |- gray
  % this maps directly into hexblack
  {0 0 0 4 -1 roll 1 exch sub 0 0}

  % |- red green blue
  % hexcyan    = 1 - red
  % hexmagenta = 1 - green
  % hexyellow  = 1 - blue
  % hexblack   = 0
  % hexorange  = 0
  % hexgreen   = 0
  {   3 -1 roll 1 exch sub
      3 -1 roll 1 exch sub
      3 -1 roll 1 exch sub
      0
      0
      0
    }

  % |- cyan magenta yellow black
  % hexcyan  = (1-CtoHG) * cyan
  % hexmagenta = (1-MtoHO) * magenta
  % hexyellow  = (1-YtoHO-YtoHG) * yellow
  % hexblack   = black
  % hexorange  = MtoHO * magenta + YtoHO * yellow
  % hexgreen   = CtoHG * cyan    + YtoHG * yellow

  {   3 index 1 //CtoHG sub mul
    3 index 1 //MtoHO sub mul
    3 index 1 //YtoHO sub //YtoHG sub dup 0 lt {pop 0.0} if mul
    4 -1 roll
    6 -1 roll //MtoHO mul 5 index //YtoHO mul add
    7 -1 roll //CtoHG mul 7 -1 roll //YtoHG mul add
    }
} cvlit
end

%eof
```

Note: In this example, as in others throughout this document, Harlequin does not recommend the exact numbers used in the redistribution as being the best available choices. These are numbers chosen empirically from experience with particular printers and inks. It is highly likely that other numbers will be better for these and other devices, media, and colorants.

Note: CIE colors are transferred to device space using an N-color CRD, which is a Harlequin extension. See Section 12 on page 33.

5.5 Calibration, color management, and profiles

You will need to address calibration and probably color management to obtain the best results from the supported output devices, but you can add these features after testing that the plugin behaves correctly in the Harlequin RIP GUI and accepts data without errors.

See Section 11 on page 31 for details of calibration and color management.

5.6 Test that the N-color plugin does what you expect

Repeat the tests that you carried out when producing a modernized plugin, as described in Section 4.5 on page 12.

Note: Carry out the tests in a new installation of the RIP. Each time that you change the plugin's `rasterFormat`, you must use a fresh SW folder. If you fail to do this, the RIP will confuse information from the old and versions of the plugin.

Follow these tests with jobs written to use the colorant aliases you have supported in your plugin.

6 PostScript LanguageLevel 3 workflows

At the time of writing in mid-1999, many jobs are still defined in terms of CMYK or device-independent colors, but some applications are able to produce jobs using DeviceN colorant definitions.

Harlequin expects various DTP and prepress applications to support DeviceN (and other features of PostScript LanguageLevel 3) in slightly different ways. As in earlier versions of the RIP, there may need to be special handling of output from some of these applications.

This section records some of the differences that may require special handling in plugins.

6.1 PageMaker 6.0 and 6.5

These versions of PageMaker use a procedure called **`sethificolor`** that is not defined in [RB3]. This procedure supports the ability of a separating application to reproduce a spot color by using contributions from more than one separation. (It is specifically intended for representing a color as a proportion of two or more HiFi colorants, but is more generally useful.) However, when the procedure is run outside the context of a separating application that understands the convention, the spot color is converted to separations in DeviceCMYK.

There is a procset that ensures that RIP version 5.1r1 and later understands the convention. The PostScript code is also applicable to Harlequin RIP version 5.1r0. See Section B.6 on page 56 for details.

7 How is PhotoInk different?

Photo-ink devices support a DeviceN output colorspace while presenting the appearance of an established colorspace (probably CMYK) to the user. This is intended to support N-color systems where some of the colorants differ from others only in density, not hue. The use of two or more intensities of related colorants removes the necessity for a single screen to reproduce the full range of tones. As a result, it is easier to avoid screening artifacts.

The screens are N-color but the calibration sets, targets, profiles, and so on are four-color (assuming that the established color space being emulated is CMYK).

For a photo-ink device, calibration in the emulated color space is likely to be more accurate because it reflects properly the way that the device will be used, especially if the delivered amounts of light and dark inks may vary. (It also reduces the materials usage and labor in calibration.)

It is easiest to explain this by considering an example for which Harlequin has implemented complete support, **`PhotoInk`**.

Using the **PhotoInk** color space available with RIP version 5.1 or later, the RIP treats the device as CMYK for the purposes of calibration and color management. It uses more than those four colorants when outputting, with a split to the extra colorants at the last stage of output (using the custom conversions procedures).

7.1 Inform the RIP

To set up a **PhotoInk** device, the procedure is similar to **Hex** (described in Section 5.1 on page 13) but with minor differences:

- Set **numFixedColorants** to N, the number of output colorants.
- Set **numAliases** to 1.

  Note: PostScript-language and PDF jobs never use the output colorants of a photo-ink system directly (unlike the **Hex** example). This means that aliases are not needed.
- Set the contents of the string **pColorantFamily** to the name **PhotoInk**.
- Set the contents of the string **pCalibrationcolorantFamily** in the **rasterFormat** to DeviceCMYK.

  This is a name, used here to allow for potential expansion when it is common to emulate more than four colors on photo-ink type devices using multiple densities of those colors.
- Use colorant names to match the supplied screens for the six-color CMYKcm system.
- Use **pCustomConversions** to distribute the colors from CMYK to CMYKcm.

7.2 Colorants

In RIP version 5.1 and later, the **PhotoInk** family uses the colorant names:

Photo Cyan
Photo Magenta
Photo Yellow

Photo Black
Photo Cyan Light
Photo Magenta Light

Note: The **`PhotoInk`** colorant family is only one possible arrangement of inks. It is easy to anticipate a number of other colorant families based on CMYK, using multiple inks for other colorants and using more than two densities of some inks. Harlequin expects to support names of the form **`PhotoInk`** *nnnn* where each of the *n* characters represents, in order, the number of inks for each of the colorants C, M, Y, and K. For example, the existing CMYKcm (or equivalently CcMmYK) family would be called **`PhotoInk 2211`**.

7.3 Custom conversions

Custom conversions provide the formula for distributing the colors of the job into the N colors of the output device. The custom conversions must perform the same two functions as in the general N-color case: manipulating the stack and color values correctly, and redistributing the color. For photo-ink systems, the redistribution is required, rather than optional, but only for the colorants represented by light and dark inks.

Note: In the following example, as in others throughout this document, Harlequin does not recommend the exact numbers used in the redistribution as being the best available choices. These are numbers chosen empirically from experience with particular printers and inks. It is highly likely that you can find other numbers that will be better for particular devices, media, and colorants.

Here is an example that performs a light/dark split for incoming data defined as CMYK or RGB (and passes Gray data only to the single black). The printer has light and dark versions of cyan and magenta inks but only one version each of yellow and black inks.

```
%!PS
% Custom conversion routines for PhotoInk printing.

3 dict begin

% The following is clearer for development but use of def
% should be avoided for released versions of this.
% The increased use of VM with def is bearable in medium term.
```

```
/PhotoInkDict currentdict def

/SplitComponent {
% Stack holds: color begin_dark end_light SplitComponent
%               light_color dark_color
  /end_light exch def
  /begin_dark exch def

  dup begin_dark le {
    end_light div % light_color
    0             % dark_color
  }{

    dup end_light le {
      dup end_light div                          % light_color
      exch begin_dark sub 1 begin_dark sub div % dark_color
    }{
      1                                          % light_color
      exch begin_dark sub 1 begin_dark sub div % dark_color
    } ifelse
  } ifelse
  2 copy add 1 gt {
    % Trim the light component if Light+Dark > 100%
    exch pop dup 1 exch sub exch
  } if
} def

{                             % start building the 3 procedures
  % |- gray
  {
    % Calculate the Black component from Gray
    0 0 0 4 -1 roll 1 exch sub 0 0
  }

  % |- red green blue
  {
    //PhotoInkDict begin

    % Calculate the Light and Dark Cyan component from Red
    3 -1 roll 1 exch sub
    0.2 0.8 //SplitComponent exec
    4 1 roll

    % Calculate the Light and Dark Magenta component from Green
    3 -1 roll 1 exch sub
    0.2 0.8 //SplitComponent exec
    4 1 roll
```

```
    % Calculate the Yellow Component from Blue
    3 -1 roll 1 exch sub 3 1 roll

    % Black Component is zero
    0 3 1 roll

    end
  }

  % |- cyan magenta yellow black
  {
    //PhotoInkDict begin

    % Calculate the Light and Dark Cyan from Cyan
    4 -1 roll 0.2 0.8 //SplitComponent exec 5 1 roll

    % Calculate the Light and Dark Magenta from Magenta
    4 -1 roll 0.2 0.8 //SplitComponent exec 5 1 roll

    end
  }
} cvlit               % end building the 3 procedures

end

% End PhotoInk custom conversions --------------------------
% end of example
```

8 What about the screens?

An N-color plugin with halftone output is likely to require a special set of screens. Some of the screens provided with the RIP can be used but others are only suitable for use with devices using four colors or fewer. You may also wish to add new screensets for use only with a specific N-color plugin.

8.1 Screen names

You must add any new screen definitions to the RIP and, usually, you will want to name the corresponding dot shapes on the menus visible to the user.

Since RIP version 4.0, **SW/Screens/sf.ps** holds the definitions of screens centrally. The **SW/Config/Screen Names** file associates definitions in **sf.ps** with the names shown in **Dot shape** menus in the Edit Style and Edit Calibration dialog boxes. [SWEXTN], Section 9.8, describes these RIP-based files and how to change them.

In RIP version 5.0 and later, a plugin can offer its own list of screens to the user, which allows it to hide inappropriate choices from the user. See Section A.2 on page 37 for details.

Here is an extract from the **Screen Names** file for a PhotoInk printer:

```
/Screening
[
  <<
    /InternalName (PhotoEuclidean)
    /ExternalName (Euclidean)
    /Type (SpotFunction)
    /Enabled true
  >>

... % other screens, renamed or added if more dot shapes wanted

]
```

Note: This example presents the screen with internal name **PhotoEuclidean** to the user with the name **Euclidean**. This simplifies the user's view of screens by providing unchanging menus for devices with different color spaces but allows different underlying definitions to be used to suit those output color spaces.

Alternatively, the plugin can suppress the user's choice of screens entirely and use screens controlled by the plugin's PostScript code. To deny the user any control of screens, in the **rasterFormat** set the field **screensRequired** to the value **screensRequired_none**, but nevertheless set the **rasterDepth** to **1**. (The screen still needs an entry in **sf.ps**.)

8.2 Provided screen definitions

Most screen sets are based around four colors. For example, the HPS system provides moiré-free screening in four-color work. When more than four colorants are to be used, there are alternative methods of avoiding moiré:

- One approach is to use stochastic screening such as HDS, adapted to suit more colors.
- Another approach is to use a conventional four-color screen set but ensure that two colors can share one screen.

RIP version 5 provides examples of both methods. There are six distinct HDS screens in each set, for both Hex and PhotoInk colorant families. For photo-ink work, the **`PhotoEuclidean`** dot shape for Photo Cyan Light is the inverse of that for (dark) Photo Cyan and the dot shape for Photo Magenta Light is the inverse of that for (dark) Photo Magenta.

Look in **`sf.ps`** for details. For example, look at **`Hds-a6`** for Hex and **`Hds-a6p`** for PhotoInk. Both these screensets provide screens for six named process colors and a seventh called Default for spot colors.

Note: For RIP versions 5.0 through 5.1r1, the six-color HDS screen caches are supplied as files outside the RIP. You must install them in the RIP, by simple copying of their folders to **`SW/Screens/`**, before you can use them.

When screens are shared, the sharing scheme is relatively obvious for photo-ink colorspaces. It would very unusual to use multiple colorants of the same hue but different densities at the same place on the page so, with some care at the transition points between using the inks of different densities, it is reasonable for these inks to share a screen. (This care is largely represented by the use of inverse versions of the same screen for the inks that may be used at a transition.)

When using a six-color HiFi family of distinct colorants, Harlequin's convention has been to use at most three colorants at any one location. While it may be a less obvious choice than in the photo-ink case, it is reasonable to make Magenta and Orange into alternatives, and also Green and Cyan.

Note: Harlequin does not recommend any particular scheme. All such screens are trial versions.

8.3 Reducing patterning with Euclidean screens and PhotoInk devices

Note: This section was added June 2000 after some problems with the trial screens supplied with the Harlequin RIP when used with PhotoInk inkjet printers.

Using a Euclidean halftone screen when printing to PhotoInk devices can produce good results if you edit the default screen definitions and take care when setting up the RIP. The following steps are required:

1. Open the file **`SW/Screens/sf.ps`**
2. Locate the section starting **`%PhotoInk Euclidean screen set`**.
3. Locate the **`/SpotFunction`** key for **`(Photo Cyan Light)`** and edit the spotfunction definition so that it matches the spot function definition for **`(Photo Cyan)`**. To do this, remove the values **`-1 mul`** preceding the bind operator.
4. Locate the **`/SpotFunction`** key for **`(Photo Magenta Light)`** and edit the spot function definition in the same way, so that it matches the spot function definition for **`(Photo Magenta)`**.
5. Save and close this file.
6. In the Harlequin RIP, open the Separations Manager dialog box, using the menu option **Color > Separations Manager**. Select a **Device** that supports PhotoInk.
7. Choose the table entry for **`PhotoInk Composite`** and click **Edit**.
8. In the Edit Style dialog box, choose **`Euclidean`** from the **Dot Shape** list and set the screen **Frequency** to 80 lpi.
9. Enable HPS by selecting the **Use Harlequin Precision Screening** check box.
10. Set the screen angles for the separations to the values in Table 0.1:

Separation	Angle
Photo Cyan	45.0
Photo Magenta	75.0

Table 0.1 Recommended angles for PhotoInk colorants

Separation	Angle
Photo Yellow	0.0
Photo Black	15.0
Photo Cyan Light	45.0
Photo Magenta Light	75.0

Table 0.1 Recommended angles for PhotoInk colorants

11. Finally, check that the following are enabled: **Override angles in job**, **Override dot shape in job**, and **Override frequency in job**.

If you have followed this procedure exactly, you should not see any patterning when using Euclidean screens for PhotoInk devices.

9 sRGB values and Roam

In **`D_GET_COLORANTS`** you had to supply sRGB values. What are these numbers?

sRGB is a shorthand way of referring to a particular scheme of calibrated RGB color definitions. See **`http://www.srgb.com/`** for more information.

- The values are used by Roam in order to display the page in its correct color.

  In RIP version 5.0 and later and when using a calibrated monitor, Roam is colorimetric for one colorant, and approximately colorimetric for combinations of two or three colorants. If the monitor is uncalibrated, the results are not colorimetric but should still be an improvement upon Roam in Harlequin RIP 4.x.

- The values are also returned in the colorants list in the **`pageHeader`** so that the plugin can access them if it needs to know the colorimetry of the colorants.

10 The All separation and ink delivery

When describing the Separation color space, [RB3] section 4.8 defines the special colorant name **`All`**. When using **`All`**, the PostScript-language rule says "painting operators apply tint values to all available colorants at once".

Unfortunately, doing this can exceed the ability of a composite printer to produce clean images by delivering too much ink. This can be a problem even when these marks are confined to marginalia such as labeling text, register marks, and step wedges. Finding an acceptable compromise for this rule becomes harder as N increases.

Even for CMY or CMYK printers it is possible to deliver more colorant than the receiving medium can accept. Using liquid inks, the limit for the maximum amount of colorant may be around 280% and for laser printers or copiers using dry toner the limit may be as low as 220%. Applying more colorant can lead to incomplete drying or media distortion, such as warping or stretching, in liquid ink systems or toner flaking in laser printers.

The problem with **`/All`** increases when there are more colorants. A lower tint value multiplied by a higher value of N can still break the limit.

There is a Harlequin RIP extension to make the **`/All`** separation produce only selected colorants. It is the page device key **`/ConvertAllSeparation`** available in RIP version 5.1 or later, and described in Section B.5 on page 55.

There are other ways of controlling the maximum amount of ink delivered to the media but they are dependent on the source of the data and on color management options.

With color management, the total output for various data sources can be limited by the methods described in [SWHCPS], which can be summarized as having limits embedded in device-link profiles or available in the user interface. See also Section 11 of this document.

11 How do I do calibration and color management?

There are two cases:

- For plugins supporting conventional or photo-ink color spaces

  The supporting files and procedures are unchanged from RIP version 4.5, except that there is provision for the calibration of spot colors. In particular, **`PhotoInk`** is exactly as CMYK.

- For HiFi and other N-color systems

  Both calibration and color management are different, as described here.

There are three areas to consider for both calibration and color management:

- Targets
- Profiles, especially **`Linear`**
- Genlin and the **`import`** file

11.1 Calibration

N-color calibration raises some new issues.

First, there are potentially more process colors to print on the target. Harlequin provides various calibration targets that can accommodate up to seven colors, including a version that supports the CMYKOG colorants of the Hex family plus one spot color. If you wish to write your own targets, see [SWPLUGIN], Appendix D, for the features newly supported in RIP version 5.1.

Also, which optical filter should a densitometer use when reading patches printed in Hex Orange or Hex Green colorants? Some manufacturers have provided special filters for these colors. In other densitometers or when calibrating strips printed in a spot color, the user may need to choose the most suitable filter from those available on the densitometer.

Harlequin format profiles have fields that allow specification of the filter for each colorant, either naming a specific filter or requesting that Genlin offer the user a menu of the available filters on the densitometer in use. Harlequin format profiles can also specify arbitrary color channels as well as process colors.

If the page setup does not use color management, a calibration target is printed using calibration aim values held in a **`Linear`** profile. You should provide a device-specific profile wherever possible but there is a system default **`Linear`** profile supplied with the RIP. However, using the system default **`Linear`** profile is likely to produce disappointing results, especially on highly non-linear devices such as inkjet printers.

See Section B.3 on page 47 for details of calibration procedures and the import file format.

The Harlequin RIP supports a new, dictionary-based, form of **`setcalibration`**, a Harlequin extension to the PostScript language. The new form is well suited to N-color calibration. See Section B.1 on page 38.

11.2 Color management

It may help to set out a reasonable order of doing the work that leads to a color-managed output device.

1. Produce output that is reasonably close to a typical printing press in the individual output channels. Treat each colorant separately at this stage.
2. Manipulate the curves for the channels relative to each other so that the components mix to produce a neutral tone balance
3. Encapsulate the data that you have gathered in steps 1 and 2 in the calibration as a *reference state*. You will require a different reference state for each combination of paper and inks; the use of transparency media or different screening schemes may also call for separate reference states. (Harlequin has a great deal of experience in this area and, while the procedure is too complex to give here, this experience is available to Harlequin RIP OEMs in various ways.)
4. Make an ICC-format profile using any of the third-party tools, using the reference state you have just generated in step 3 when printing the targets required by the specific tool you are using.
5. Import the ICC profile into the RIP to form a Harlequin-format profile.

Once you have a suitable profile, its use is similar to the procedures described for Harlequin Color Production Systems in [SWHCPS].

Note: There is no way to delete an ICC profile from the RIP once it has been installed. The only way to correct the effect of installing a faulty profile is to re-import a corrected version of the profile, which will overwrite the profile imported earlier.

Note: There are other device imaging characteristics than color fidelity to consider when generating profiles. One of these is the maximum amount of ink that the device can deliver on an area or pixel basis. Another is the presence or not of black in highlights.

12 Hi Fi extension to CRD

[RB3] does not specify any method to convert from device-independent colors (XYZ) to the DeviceN space of an N-color device. Using a four-color output color rendering dictionary (CRD) would produce CMYK which would then be processed by the CMYK to DeviceN custom conversion with results that are inevitably non-colorimetric.

Therefore, Harlequin has implemented an extension to the CRD mechanism.

The PostScript LanguageLevel 3 convention is that the ordering of outputs from the RenderTable in the color rendering dictionary (CRD) must match the ordering of the DeviceRGB or DeviceCMYK color space. The RIP supports a space defined as an array of color names and this allows both DeviceN output, or a subset of DeviceN, and in an arbitrary order.

See Section B.2 on page 43 for details.

Note: There is no N-color input to the CRD; you supply XYZ colors.

12.1 Importing ICC profiles

Harlequin supports the import of CMYK and RGB ICC profiles. There is currently no definition of N-color profiles published by the ICC, for input or output. Harlequin expects to support such a standard when it is agreed and published. This standard is due very soon (from the time of writing, May 1999).

The ICC web site is at **`http://www.color.org/`** and contains published specifications and member lists. It contains little further information for non-members; for example, proposed specifications are not available.

When they become available N-color ICC profiles may become a valuable source of data for generating CRDs.

Appendix A

Supplement to the Plugin Kit Guide

The following items are not documented in the Plugin Kit Guide for the Harlequin RIP version 5.1:

- Certain additions to the plugin interface
- The ability to define screen names for each plugin
- The N-color cases of frame/band interleaving

A.1 Additions for plugin interface version 16

The plugin interface version was raised to 16, with the release of the Harlequin RIP version 5.1. There are two additions to the **`rasterFormat`**.

`pCalibrationColorantFamily` *Wr*

Note: Applies only to RIP version 5.1 and later.

Type: **`uint8*`**

A string of up to 64 characters which names the colorant family to use for calibration and color management. For photo-ink devices, this declared colorant family can be different from the actual colorant family of the output device.

At the RIP version 5.1, the only expected values are **`DeviceCMYK`**, **`DeviceRGB`**, **`DeviceGray`**, **`null`**, or an empty string.

If the device uses a photo-ink colorant family, choose the value for the related color space. (CMYK photo-ink devices are the best known examples in the middle of 1999.)

If the output device is not photo-ink, set the string to the same as **`pColorantFamily`**. Setting the string to **`null`** or an empty string is equivalent.

Warning: This is a pointer to an existing string for the plugin to fill in. Beware of overwriting this pointer with one of your own if you are doing static structure assignment.

`calibrationAsPress` *Rw*

Type: **`int32`**

Note: This item was named **`calibrationIsSeparating`** in the RIP version 5.1r0. The name shown is used for version 5.1r1 and later.

Some devices, for example, the CIP3 file device, define composite raster formats but must emulate a separating output device as closely as possible. As a result, the user should be able to calibrate and color manage the device as though it were a separating device. Set this field to **`TRUE`** (or 1) if this is the case. Set this field to **`FALSE`** (zero) if this ability is not required.

This key affects the Harlequin RIP GUI only, though it would be reasonable for other implementations of calibration to behave similarly. The GUI changes are:

- **Intended press** and **Actual press** in the Page Setup dialog box are available when the flag is **`TRUE`**.
- In the Color Setup Manager, if this flag is **`TRUE`** for all raster formats supporting a particular colorant family, then that colorant family does not appear in the **Colorspace** menu.
- In the Calibration Manager and subsidiary dialog boxes, there is expected to be an effect in RIP version 5.1r1 or later when separating and non-separating styles become treated differently.

A.2 Screen Names specific to a plugin

You can use a file (or several) in the plugin's **`Screen Names`** folder to define screen names unique to the plugin, as seen by the user. Using such files, you can add screens to those provided by the RIP, and you can also choose to hide or rename other screens as appropriate to a device or plugin.

The name you give files in this folder is important:

- Name the file for the relevant device type if the screen names it contains are specific to one device.
- Name the file **`Screen Names`** if the screen names can be used by several devices. (You can also use this name if the plugin supports only one device type.)

You can use both types of naming. (If you do not provide any screen naming files for the plugin, the **Dot shape** menus display screen names provided by the RIP.)

The format of screen names files placed within the plugin folder is the same as that documented for **`SW/Config/Screen Names`** in [SWEXTN] Section 9.8. You must list all the screen names that you wish to appear in the menus for a device or plugin in one file; there is no merging between files.

The following list is a summary of the description above. Each time the RIP has to present a menu of dot shapes to the user, the RIP checks the plugin and device type and uses the file found first when searching the following locations in this order:

```
SW/devices/plugin_name/Screen Names/device_type
SW/devices/plugin_name/Screen Names/Screen Names
SW/Config/Screen Names
```

A.3 N-color frame and band interleaving

The diagrams of data arrangements in memory, shown in [SWPLUGIN], Fig 3.1, need expanding to illustrate the general N-color cases.

There are some related changes required in [SWEXTN], listed in Section B.4 on page 54 of this document.

Appendix B

Supplement to Using Harlequin RIP Extensions

There are several new items that do not appear in the *Using the Harlequin RIP Extensions* manual. These are:

- N-color form of **`setcalibration`** operator
- CRD definition supporting N-color output
- Import calibration (Genlin) file formats
- Modified support for raster formats previously defined by **`/Tones`**
- Page device key to handle the **`/All`** separation

B.1 N-color form of setcalibration operator

Note: This form of **`setcalibration`** was added for RIP version 5.0. See [SWEXTN], section 9.9, "Calibration", for details of the two earlier array-based forms, which continue to be supported.

The array style of **`setcalibration`** is not adequate for N-color, because the colorants are only defined by convention for CMYK, RGB, and Gray — using a scheme that is not easily extended to arbitrary colorants.

RIP version 5.0 and later supports a third form of **`setcalibration`** which is modeled on the type 5 halftone dictionary: that is, it contains a sequence of keys, with the names of each colorant entry, and associated dictionaries. There may be a **`/Default`** colorant key for use with any colorants not otherwise named; see Section B.1.2 for a description of what happens when this key is not present.

Here is an example for a Hex (six-color CMYKOG) colorant family. See sections B.1.1 and B.1.2 for the contents of the subsidiary dictionaries, shown here by **<<...>>** :

```
% Dictionary-based calibration of Hex printing

<<
  /CalibrationType                5

  (Hex Cyan) cvn                  << ... >>
  (Hex Magenta) cvn               << ... >>
  (Hex Yellow) cvn                << ... >>
  (Hex Black) cvn                 << ... >>
  /Default                       << ... >>
  (Hex Orange) cvn                << ... >>
  (Hex Orange) cvn                << ... >>

  /ForceSolids                    boolean   % default false
  /NegativePrint                  boolean   % default false
  /WarningsCriteria               << ... >>
>>
1183615869 internaldict /setcalibration get exec
```

Note: When using the GUI version of the RIP and its built-in capabilities, it should never be necessary to construct this dictionary; all the necessary information is created based on user actions in the Calibration Manager and Edit Calibration dialog boxes. You may need to use this information if you are using a Harlequin RIP without a GUI (for example, SOAR) or if you wish to override the calibration that the GUI version would provide.

Notes on the keys:

The **/CalibrationType** key is required, and always has value **5** in dictionaries created by the Harlequin RIP. (The value could be **1**, for a single colorant, but Harlequin software does not use this option.)

The keys for colorants, including **Default**, can be names as shown or strings. See Section B.1.1 for details of the subdictionaries.

ForceSolids and **NegativePrint** are optional, and can also appear in the colorant subdictionaries, which is their preferred location. If present here, the values provide defaults, overridden by any values given in the colorant subdictionaries. See Section B.1.1 for details.

WarningsCriteria is optional, and can be used to specify when there will be warnings about use of unsuitable calibration sets. See Section B.1.2 on page 42 for details of the subdictionary.

B.1.1 Colorant subdictionaries

The values associated with each colorant are dictionaries containing the constituent calibration sets for the colorant. There may be a **Default** colorant dictionary which will be used for all colorants not given explicitly named dictionaries. The possible keys in each colorant dictionary are:

```
/colorant_key <<
  /CalibrationType                1
  /DeviceCurve                    array    % default [ ]
  /ToneCurve                      array    % default [ ]
  /IntendedPressCurve             array     % default [ ]
  /ActualPressCurve               array     % default [ ]
  /ForceSolids                    boolean   % default false
  /NegativePrint                  boolean   % default false
>>
```

For each colorant, the curves are applied in this order:

1. IntendedPressCurve (as a backwards transform)
2. ActualPressCurve
3. ToneCurve (as a backwards transform in RIP 5.1r1 and later, but applied normally in versions 5.0 through 5.1r0)
4. DeviceCurve

Note: A curve is used as a normal transform if a value on the measured axis is used to derive the device code that produces that value. [SWEXTN], Section 9.9.1 explains this usage. A curve is used as a *backwards transform* if it is used in the opposite sense, perhaps to remove a pre-applied calibration; you can view this as using the device code to derive the desired (measured) value — a backwards progress from one axis to the curve to the other axis — or as a "flip" of the curve about a straight line from 0.0, 0.0 to 1.0, 1.0.

See also the comments on ordering in [SWEXTN], Section 9.9.5.

Notes on the keys:

Only the **`CalibrationType`** key is required, and it must have value **`1`**. If there are no curve-related keys, a colorant is associated with no calibration sets (and this implies linear calibration).

The arrays in this dictionary are in the same form as the calibration arrays used in RIP version 4, and are derived from calibration sets in the GUI version of the Harlequin RIP. Each array contains a series of points (where each point is an input, output pair of numbers) and must be zero length (linear calibration) or contain at least two calibration points (four numbers). All the curves consist of Nominal Value / Normalized Device Code (SNV / NDC) pairs.

Note: The HqnCalibrate procset, which handles calibration, derives data in the two press curves, **`IntendedPressCurve`** and **`ActualPressCurve`**, from two sources. Without color management active, we use "Normalized DotGain" / NDC pairs, that is, the DotGain percentage values are divided by 100 (to bring values into the range 0 through 1). With color management, the **`ActualPressCurve`** uses SNV/NDC pairs while the **`IntendedPressCurve`** is not present. (The function of **`IntendedPressCurve`**, and more, is performed by colorimetric data in the color management input profiles). Other uses of **`setcalibration`** should follow similar rules.

NDC values in the arrays are expected to be in the range 0 through 1 and to be strongly monotonic in each curve: that is, strictly increasing or decreasing with no values equal.

SNV values in the arrays are expected to overlap the range of 0 through 1 and be weakly monotonic in each curve: that is, strictly increasing or decreasing but allowing consecutive values to be equal. This definition allows some values to be outside the range of 0 through 1 but only that part of the range that lies in the range 0 through 1 will be used by **`setcalibration`**.

Set **`ForceSolids`** to **`true`** if 100% colorant is to be preserved.

Set **`NegativePrint`** to **`true`** if the **`DeviceCurve`** is for a negative. It has the effect of inverting the measurement data for the **`DeviceCurve`** but has no effect for any of the other curves.

B.1.2 WarningsCriteria subdictionary

Note: This key, **WarningsCriteria**, was added for RIP version 5.0.

There is an optional key, **WarningsCriteria**, in the top level of the dictionary operand to **setcalibration**, introducing a subdictionary of this form:

```
/WarningsCriteria <<
  /MissingCalibrationAbort  boolean
  /DeviceCurve              << ... >>
  /ToneCurve                << ... >>
  /IntendedPressCurve       << ... >>
  /ActualPressCurve         << ... >>
>>
```

The **WarningsCriteria** subdictionaries have two effects:

- When present, each subdictionary sets a range of applicability for calibration sets.
- When absent, each key/subdictionary pair also has an effect on warnings about the use of substitute calibrations for spot colors.

The second effect needs some explanation. Remember that any colorant without a specific calibration, typically a spot color not represented in the calibration set, inherits the calibration for **/Default**. If there is no calibration for **/Default** in a curve (device curve, tone curve, and so on) then the colorant will attempt to use the calibration for **Black** (and give an appropriate warning) or detect that there is no **Black** and assume a linear calibration (and give a different warning). For example, if there is no **DeviceCurve** key in the **WarningsCriteria** dictionary, then there will be no warnings about the use of the **Black** or linear calibration *for the device curve*.

Notes on the keys:

The **MissingCalibrationAbort** key is required (if the **WarningsCriteria** dictionary exists). Set to **true** to abort jobs where there is a missing calibration. Set to **false** to continue the job after issuing a warning.

All of the keys introducing subdictionaries in **WarningsCriteria** are optional.

B.1.3 Curve-related subdictionaries

If present, each subdictionary of **`WarningsCriteria`** accepts the optional keys shown in the following example, where `/xxxCurve` represents one of the keys with a subdictionary:

```
/xxxCurve <<
  /HWResolution   [xRes yRes]
  /Exposure       exp
  /NegativePrint  boolean
  /HalftoneName   (int_name)
  /Frequency      [low  high]
>>
```

Notes on the keys:

All keys are optional. An absent key means that any match is allowed.

`HWResolution`, **`Exposure`**, and **`NegativePrint`** specify the corresponding values from **`currentpagedevice`**, that must match at the time all screens are installed. *xRes* and *yRes* can be integers or reals (in dots per inch), but the exposure value *exp* must be an integer (as shown in the RIP user interface), which has no units and which has a value range and meaning that vary from device to device.

`HalftoneName` specifies a single (internal) name that must match all screens used. Some examples are: Euclidean, Round, or Hds-a6. The screen must have an external name visible in the **Dot Shape** menu of the edit calibration dialog box.

`Frequency` specifies a low and high value (in lines per inch) for the range allowed by all screens used.

B.2 N-color CRD

Note: This Harlequin extension, introduced for RIP version 5.0, provides a colorimetric method of converting colors defined in an XYZ-based color space to a DeviceN output color space.

The standard **RenderTable** is defined in [RB3], section 7.1, page 467 as:

> An array of the form **[N_A N_B N_C table m T_1 T_2 … T_m]** defining a three-dimensional lookup table that maps colors in render color space into device color space.

For example, one might have:

```
<<
  /ColorRenderingType 1
  ...
  /RenderTable [
    33 33 33 [ <...> <...> ... <...> ]
    4
    { } { } { } { }
  ]
>> setcolorrendering
```

[RB3] says that the value of *m* must be 3 or 4 but the Harlequin RIP extends the three- and four-color output from the CRD to any number of colorants. However, to do this, it must be possible to tie the colorants that those outputs represent to the colorants of the device; and also to convert to other device color spaces if the colorants do not match (in the same way as the tint transform required for a **DeviceN** space given to **setcolorspace**).

To do this, the Harlequin extension allows *m* in the **RenderTable** (element index 4 of the array, counting from zero) to be a name or an array which specifies a color space. This name or array implicitly specifies the number *m*, as well as supplying the other information needed. There are two consequential changes in the rest of the render table:

- There must be the appropriate number of procedures at the end of the render table for the color space provided. (The total length of the **RenderTable** array is *m*+5.)
- The strings in the render table contain one byte for each colorant, the obvious extension from the ordinary color rendering dictionary.

For example one can have:

```
<<
  /ColorRenderingType 1
  ...
  /RenderTable [
    33 33 33 [ <...> <...> ... <...> ]
    /DeviceCMYK
    { } { } { } { }
  ]
>> setcolorrendering
```

This is exactly equivalent to the first example. More usefully, one can have, for example:

```
<<
  /ColorRenderingType 1
  ...
  /RenderTable [
    33 33 33 [ <...> <...> ... <...> ]
    [/DeviceN [/HexC /HexM /HexY /HexK /HexO /HexG]
              /DeviceCMYK { pop pop } bind ]
    { } { } { } { } { } { }
  ]
>> setcolorrendering
```

This example means that the CRD produces the six colorants named in the color space as its output: **`HexC`** and so on. There are two cases, each with its own processing path:

- If these output colorants appear in the final device space, there is the obvious mapping to them. This happens even when the CRD's output colorants form a subset or a reordering of the colorants in the device space. (It may be convenient for the CRD to produce a subset when the device space is made up of a standard process family plus spot colors.)
- If one or more of the CRD's output colorants are *not* present in the final device space then the tint transform procedure is used. In this example the conversion is to DeviceCMYK, using a simplistic conversion that simply discards the last two color values of the six. This is just like normal processing of a DeviceN color space.

In either path, color processing proceeds from then on in exactly the same way as for DeviceN. Figure B.1, shows how this Harlequin extension modifies the

possible color rendering paths compared to those shown in [RB3], Figure 4.6, *Color Rendering*.

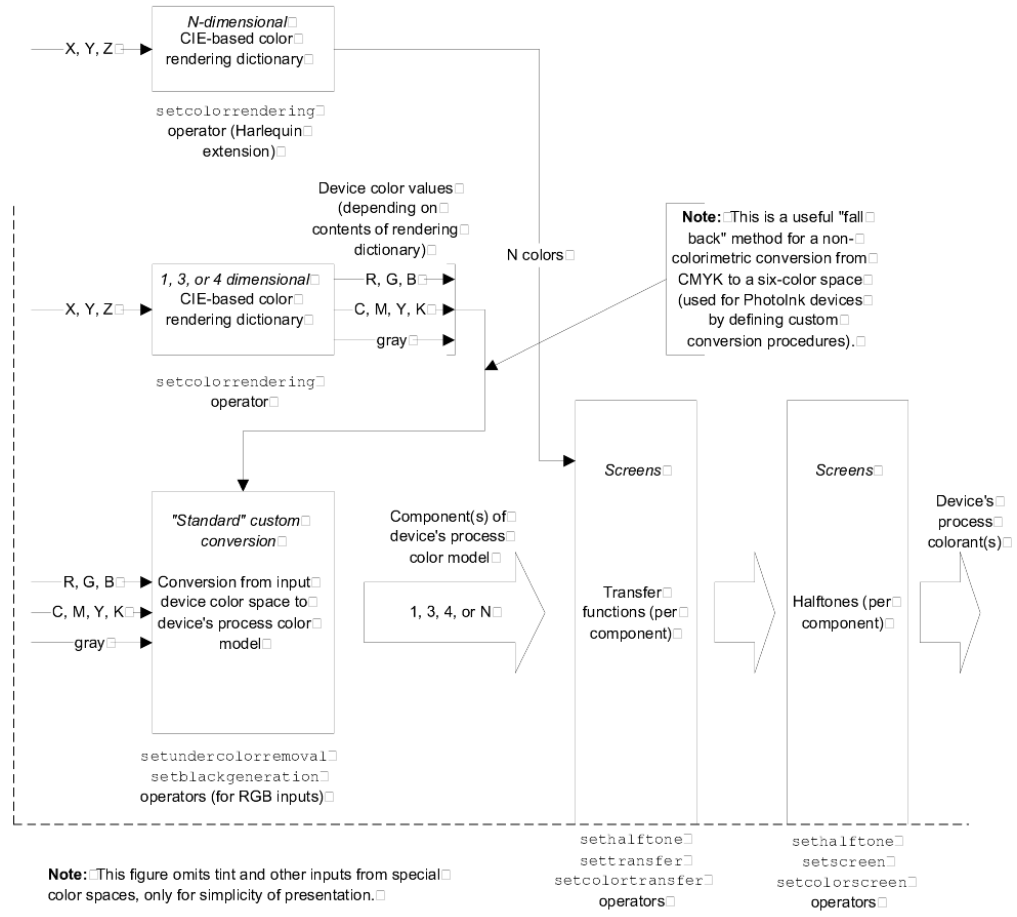


Figure B.1 Color rendering, as modified by the setcolorrendering extension

B.3 Calibration and Import (Genlin) file formats

Note: This section modifies the information given in [SWEXTN], section 9.9.7, "Importing calibration tables".

At RIP version 5.0 and later, calibration is a more complex affair than with earlier versions:

- There are new colorspaces with more process colors. The standard targets accommodate up to seven colors. (The extra colors can be new process colors, spot colors, or a mixture of both.)
- It is possible to print and measure calibration targets for spot colors.
- It may be possible to use different purpose-designed optical filters for the new process colors but it may be necessary for the user to choose the most suitable of the filters available on older densitometers.
- It is possible to print several calibration targets and then measure them with Genlin in an order chosen by the user. Also the user is able to choose which strips to measure when the target contains several channels.
- When importing the measured data into the RIP, the user can choose to import only some channels and to reassign measured channels to other colorants. For example, a typical use is to use the Black measurement data for spot colors. Other reassignments or copying between channels may be useful and the user is free to make such changes.

Harlequin RIP 5.x and Genlin are designed to make the complete calibration process as simple and error-free as possible. To do this, the RIP stores information about the target and the device profile at the time of making the target. This stored information is available to Genlin at the time of measuring and to the RIP at the time of importing data.

The end result is that Genlin measures the data using a measurement system available to both the measuring instrument and the profile. In the RIP there is a check that the user is importing the measurement data for the correct device and profile, and using the correct measurement system. This eliminates some common errors possible with RIP 4.x.

Note: There were some problems with profiles that named the Status-T measurement system in various incompatible ways.

Note: The same Genlin executables can be used with RIP version 4.5 RIP where they exhibit the same behavior, and generate the same format of files, as expected by that version of the RIP. Genlin examines the **SW** folder to determine the version of the RIP, and behaves in a compatible manner.

B.3.1 Calibration procedure

The outline procedure for printing a target, using Genlin, and importing the data into RIP version 5.x is this:

1. Choose **Output > Print Calibration**. The Print Calibration dialog box offers a list of page setups (as in RIP version 4.x) and a new **Print for**: menu offering some of the options:

   - **`Process Colors only`**
   - **`Spot Colors only`**
   - **`Process & Spot Colors`**
   - **`Monochrome only`**

   The options offered depend on the selections in the **From Page Setups** list and the separations style chosen in each of those page setups. The spot color options appear only if the page setup defines spot colors.

   The **`Monochrome only`** option supports calibration of a monochrome device such as an imagesetter. Choose this option to print a single calibration strip for a page setup that normally generates separations. (This ability removes any need to produce a page setup used only for calibration.)

   The RIP prints calibration targets using calibration sets that depend on the type of target required and the page setup. For calibrated targets, the RIP always uses the calibration from the page setup. For uncalibrated targets, the Harlequin RIP searches a number of places in this order:

   - With color management active, the RIP uses the **`DefaultCurve`** from the (output) profile in the color setup.
   - With color management inactive, the RIP uses the **`DefaultCurve`** from the profile in the selected calibration set. **Note:** The use of this location was introduced in version 5.1r0.
   - If the calibration is set to **`(None)`**, the RIP uses the **`DefaultCurve`** from the **`Linear`** profile. The **`Linear`** profile used may be one specific to the device or the system default for the Harlequin RIP installation if not found in a device-specific location.

   The RIP always disables color management when printing calibration

targets.

2. Click the button that prints the target(s) you want. The RIP creates a page buffer of the calibration target. When printed, this becomes a hard-copy target (or a file in a format such as TIFF). The Harlequin RIP also creates entries that help Genlin associate profiles with targets. These entries are in these places:

   - **`SW/caldata/trgetinf`**
     This file records the next available file names for files in the **`trgetpnd`** and **`trgetmap`** folders. The file names are integers counting up from 1.
   - **`SW/caldata/trgetpnd/`***target_number*
     Each numbered file describes a printed target briefly, with a summary of device, profile, and measurement conditions. It also references a file in the **`trgetmap`** folder to describe the target fully.
   - **`SW/caldata/trgetmap/`***map_number*
     Each file describes a different target format, which may be used by several of the files in the **`trgetpnd`** folder. A new file is required only if a new target is used)

   The RIP prints the *target_number* with the label **`Reference Number`** on the target and uses the next available integer for the next target.

3. When you start Genlin, choose **File > Configure** to check that Genlin is set up for the correct RIP, identified by the path to its SW folder, and measuring instrument. Once you have done this, choose **File > Read Target**. In the Choose Reference Number dialog box, choose one of the numbers to select the target you wish to measure and click **OK**.

4. Choose the channels that you wish to measure. Measure the target with a manual or automatic densitometer, following the prompts given by Genlin or a display on the densitometer. (You may need to trim some targets into smaller pieces to make all strips readable with a strip-reading densitometer.)

5. Genlin produces the measured data in the file:

   - `SW/caldata/import`

   This file contains labels identifying the device, target and profile used, and the number of colorants for which it contains measurement data. Then, for each colorant represented in the file, it contains labels giving the colorant name, the measurement system used, the densitometer filter, the number of data readings and the actual measurement data.

   See Section B.3.2 on page 52 for details of the file format.

6. In the Harlequin RIP, read the data. To do this, choose **Output > Calibration Manager**. In the Calibration Manager dialog box, select the relevant **Device** and **Color Space**, then click **New**, **Edit from uncalibrated target**, or **Edit from calibrated target** to enter the Edit Calibration dialog box (which will have a title bar corresponding to the button you clicked). Click **Import** to read the data from the file:

   - `SW/caldata/import`

   The RIP may detect a mismatch between the expected device, profile, or measurement system and display a warning dialog box. Click **Yes**, to continue, only if you are certain that the mismatch is harmless.

7. The RIP displays the Import Measurements dialog box, as described in [SWOEM], section 11.12.7. This allows you to choose which channels to read from the file, and where to use the data for the channels known in the Edit Calibration dialog box. Click **Import**.

8. Once the data is read, the RIP converts all values to the measurement system shown in the **Measurements as** menu. (You can change to the measurement system used by the densitometer if you want to see the measured values.) Check that all channels have reasonable curves, name the calibration set in the **Name** field, and click **OK** to close the Edit Calibration dialog box.

9. Repeat steps 3 through 8 for as many targets as you wish to measure. Click **OK** to close the Calibration Manager dialog box.

10. If you are sure that you have measured all the data you require, you can use **File > Purge** to remove all pending target information.

Notes:

- There are two formats of **DeviceGray** targets: a format using two strips of density patches, for manual densitometers; and a format using a single strip, suited to strip-reading densitometers. (All other targets are compatible with both types of densitometers.)

  The RIP uses the target files held in the folder:

  - **SW/Config/Targets**

  As shipped, the **DeviceGray** file held in this folder is suitable for manual densitometers only. The subfolder **xrite** holds two alternative versions of the **DeviceGray** target, for DTP*nnn* series strip-reading densitometers, such as the DTP41. These alternatives are equivalent except that one (**DeviceGraySmall**) produces a more compact calibration strip. To use one of these versions, rename the **DeviceGray** file shipped in the **Targets** folder (or copy it to a subfolder) then copy the file you wish to use to the **Targets** folder, renaming this copy to **DeviceGray** if needed.

- The profiles must match at all stages for the results to be meaningful. This means that any calibration set you import after a warning of a mismatch is suspect.

B.3.2 Harlequin RIP import file format

The **SW/caldata/import** file has this format:

```
#Device: Stylus 9000
#Profile: 9000-Gloss-720-a
#Target: Epson 9000 4-Color target
#Colorants: 4

#Colorant: Cyan
#Measurement System: Status T (X-Rite)
#Filter: Cyan
#Readings: 18
"C100",2.180000
...                 % omitted data
"C0",0.050000
```

```
#Colorant: Magenta
#Measurement System: Status T (X-Rite)
#Filter: Magenta
#Readings: 18
"M100",1.740000
...                 % omitted data
"M0",0.050000

#Colorant: Yellow
#Measurement System: Status T (X-Rite)
#Filter: Yellow
#Readings: 18
"Y100",1.220000
...                 % omitted data
"Y0",0.030000

#Colorant: Black
#Measurement System: Status T (X-Rite)
#Filter: Visual
#Readings: 18
"K100",2.150000
...                 % omitted data
"K0",0.050000
```

This fragment shows the labels present in the import file. The individual labels are:

Device	The name of the device used to generate the target from which this data was measured. For example: **`Stylus 9000`**
Profile:	This is name of the profile used when preparing the target. For example: **`Linear`**
Target	String. For example: **`1-color X-Rite-DTP-compatible`**
Colorants	The number of colorants for which there is data in the import file. The first of these colorants follows after a blank line. For example: **`4`**

The sections of the import file for each colorant are as follows:

Colorant	The name of the colorant. For example: **`Gray`**, **`Cyan`**, or **`Hex Cyan`**.

Measurement System:
: This is the measurement system used by the densitometer. For example: **`Positive % Dot`**.

Filter
: This is the optical filter actually used by the densitometer for this colorant. (The profile might have specified **`Default`**, meaning that Genlin presented the user with a list for choice.) For example: **`Visual`**.

Readings
: This integer is the number of data points for this colorant. There will be this many lines following to hold the data. For example: **`21`**.

<Data values>
: The measured data for each patch on the calibration target. Each line is of the form
`"C100",2.180000`

In this example:
`C100`, omitting the quotation marks, is the label of the patch on the target and the data value field in the Edit Calibration dialog box. The RIP can import the data only if the patch labels in this file match exactly in wording and ordering with the labels for the data fields in the dialog box.
`,` (comma) with or without surrounding spaces is the separator.
`2.180000` is the value in the measurement system declared for this colorant.

If there is another colorant, its labels will start after another blank line.

B.4 Modified support for raster formats

[SWEXTN], section 6.2, describes the raster formats supported in version 4.x.

There are some significant additions for version 5.0 and later:

- Nx1 and Nx8 band-interleaved output.
- Nx1 and Nx8 frame-interleaved output.

Note: Frame-interleaved support was introduced in RIP version 4.0 but not documented in the same place and manner.

Some uncommon forms of raster formats are not supported in version 5.0 and later, but (so far) remain without comment in [SWEXTN]. These are:

- 6 levels each of Red – Green – Blue.

  This was expressed by negative numbers in the Tones array. For example:

  ```
  << /Tones [/Red -6 /Green -6 /Blue -6] >> setpagedevice
  ```

- 3 bits Red, 3 bits Green, and 2 bits Blue; pixel interleaved (packed into an 8-bit pixel), with the color components in any order.
- An undocumented format of 2 bits Cyan, 2 bits Magenta, 2 bits Yellow, and 2 bits Black, pixel interleaved (packed into an 8-bit pixel), with the color components in any order.

Where one of these unsupported formats is required for output, Harlequin's recommendation is that your plugin should use a supported format for RIP version 5.1 for data delivery from the RIP, and then the plugin can reduce the precision for each color and repack the data to reproduce the required format.

B.5 Page device key to handle the All separation

There is a Harlequin RIP extension to make the **`/All`** separation produce only selected colorants, which can help avoid delivering too much colorant to composite printers. In the RIP 5.1 or later, you can use this in the plugin PostScript code, typically in the **`Setup`** file.

For example:

```
<< /ConvertAllSeparation /Black >> setpagedevice
```

This page device key can have the values **`/Black`**, **`/DeviceCMYK`**, or **`/All`**. If the key is not set, the default is **`/All`** or the equivalent **`null`**.

Using the value **`/Black`** means that **`/All`** marks in the black separation only, using **`/DeviceCMYK`** means marking all four separations in that color space, and using **`/All`** means following the standard PostScript-language rule and marking with all available colorants.

B.6 HqnHiFi procset and spot color support

At version 5.1r1 and later, there is a new procset, HqnHiFi, which may be called from the plugin PostScript code of plugins driving HiFi output devices. It provides procedures that support the ability of a separating application to reproduce a spot color by using contributions from more than one HiFi separation.

The HqnHiFi procset contains two procedures:

`InstallHiFi` Defines **`sethificolor`** in **`userdict`**. This means that spot colors from PageMaker 6.0 and 6.5 that have Hexachrome equivalents will use those equivalents rather than the CMYK ones when printed on a printer supporting an appropriate color set.

Note: The Harlequin RIP installation must be aware of the names Hexachrome Orange, Hexachrome Green, Hexachrome Yellow, Hexachrome Black, Hexachrome Magenta, and Hexachrome Cyan. (Typically, the output plugin for the HiFi device defines these colorant names in a relevant **`rasterFormat`** and installing the plugin declares the names to the RIP.)

`OmitBlanks` Prevents the production of blank separations if called together with **`InstallHiFi`**. Without a call to **`OmitBlanks`**, if **`InstallHiFi`** is used with a device and colorant set combination that does not include the Hexachrome names (for example, a CMYK device) and if **`(Other colors in job)`** is set to **`Yes`** in the **`Print?`** column of the Edit Style dialog box then blank separations will be produced for the six Hexachrome inks. To ensure that these separations are not produced, call **`OmitBlanks`**.

For example, to call both procedures in the procset, this code is sufficient:

```
/HqnHiFi /ProcSet findresource
dup
/OmitBlanks get exec
/InstallHiFi get exec
```

Appendix C

External references

The following books are valuable sources of supporting information.

[RB3] *PostScript Language Reference Manual, 3rd Edition* (the "Red Book"), Addison Wesley.

[HQEXTN] *Using Harlequin RIP Extensions: Guide for OEMs*, edition 5.5. Global Graphics Software Limited.

[HQHCPS] *Harlequin Color Production Systems User's Guide*, edition 5.5. Global Graphics Software Limited.

[HQOEM] *Harlequin RIP OEM Manual*, edition 5.5. Global Graphics Software Limited.

[HQPLUGIN] *Harlequin RIP Plugin Kit: Guide for OEMs,* edition 5.5. Global Graphics Software Limited.

There is also a draft Harlequin document describing the calibration-related portions of Harlequin profiles. (*Harlequin RIP: Profiles and Calibration*, June 2001).

Change history		
v 1.0	1999.05.13	First issued DRAFT, for ScriptWorks 5.1
v 1.1	1999.08.18	Corrections for ScriptWorks 5.1r1
v 1.2	2000.06.26	Additions for PhotoInk and Euclidean screening, for ScriptWorks 5.3

Change history		
v 1.3	2001.06.18	Updated cover page and copyright page. Removed references to ScriptWorks and replaced with Harlequin RIP. Updated Appendix C: External References