The PrintTicket device

Technical Note Hqn088

March 2013

1 Introduction

To enable 3rd party developers to write their own device handling PrintTickets, the RIP provides an interface to get hold of the PrintTicket XML and supply PostScript language code to configure the RIP according to the merged and validated PrintTicket parameters. The interface is built on the PostScript language device type interface. For more information see the Device specifications chapter of the Harlequin Host Renderer Developer's Guide.

The PrintTicket and application page size details are written to the device as XML, so the PrintTicket device needs to include an XML parser. The example PrintTicket device uses the Expat parser.

The PrintTicket device must provide the default PrintTicket settings, and implement the merge and validate process required to determine the effective print settings in response to processing a PrintTicket.

2 Example implementation

An example implementation of the XPS PrintTicket device can be found in the Harlequin Host Renderer and Embedded SDKs in the following location:

\HHR_XXrX\lib\ptdev\src\ptdev.c

3 PrintTicket XML

The RIP opens a file for writing on the PrintTicket device and then writes the content of the PrintTicket part to the file. Special file names are used to indicate the scope of the PrintTicket about to be written to the device. They are as follows:

`JS` Job scope

`DS` Document scope

`PS` Page scope

PrintTickets are written to the device in scope order, that is any Job PrintTicket will be sent before any Document PrintTicket, which will be sent before any Page PrintTicket. For example, if you see a Document PrintTicket without seeing a Job PrintTicket then no Job PrintTicket was present in the package and the Default PrintTicket should be used as the "Validated Job-level PrintTicket" when merge and validating.

4 RIP configuration

At certain points during the processing of a XPS Package the RIP will read configuration PostScript language code from the PrintTicket device. These points are:

- Start of DocumentSequence (Job)
- Start of Document
- Start of Page
- End of Page
- End of Document
- End of DocumentSequence (Job)

The configuration PostScript language code being read at a point is identified by using special file names in the same way as the PrintTicket scope is identified. The start point file names are the same as those for the PrintTicket XML at the same scope. That is, the start of `DocumentSequence` file name is `JS`, and so on.

The RIP will open the file, write any PrintTicket XML, and then read back any configuration PostScript language code having processed the PrintTicket. The file is closed when there is no more (a read returns end of file). If the RIP reads from the file without having written anything to it first, there was no PrintTicket at that scope level in the XPS package.

The special filenames for the end points are:

`JE`	Job end
`DE`	Document end
`PE`	Page end

The end point file will only be opened and read from if the start point file was successfully opened and read from, unless there was an error processing the XPS package since then. See "Error handling" on page 7 for more information.

Again, the RIP will close the file after a read returns end of file.

The RIP reading from the end point file can be taken as the end of the scope for any PrintTicket that was written to the corresponding start point file.

Opening a configuration file should always succeed even if there is no configuration PostScript language code to be interpreted. In this case, the PrintTicket device should return an end of file indication on the first read or query on the number of bytes available to be read. The RIP will then close the configuration file and continue processing the XPS package.

By default, the RIP is configured for PostScript language rendering. This means that the userspace is set up at 72 dpi with the origin at the bottom left of the raster, and the raster set up for a default page size. At its simplest, the page start configuration needs to set the page size, move the origin, flip the Y-axis and scale the userspace from 72 dpi to 96 dpi. This can only happen at page start since this is the first time the application media size is known to the PrintTicket device.

5 FixedPage details

In order to configure the RIP, some information from the XPS `FixedPage` element is also required. This is sent as XML to another file on the PrintTicket Device with a special name—`PD`. The following is an example of the XML written to the device:

```
<?xml version="1.0" encoding="utf-8"?>
<PageDetails xmlns=
"http://schemas.globalgraphics.com/xps/2005/03/pagedetails">"
  <Page Size="816,1056" BleedBox="0,0,816,1056"
  ContentBox="24,24,792,1032"/>
</PageDetails>
```

where the attribute values are the same as they are in the XPS `FixedPage` element XML.

Note: The `BleedBox` and `ContentBox` attributes are always present, even if they are not in the XPS `FixedPage` element, in which case they have default values filled-in by the RIP.

An XSD reference file for this structure is included in the distribution and can be found at:

\HHR_XXrX\doc\XSD\PageDetails.xsd

5.1 Element PageDetails

This is the root element of the `PageDetails` XML. It establishes the XML namespace. If the content of the `PageDetails` XML changes in the future, the namespace will change with an updated date in it. It has no other attributes. It has one contained child element—`Page`.

5.2 Element page

This element has three required attributes—`Size`, `Bleedbox`, and `ContentBox`. It has no child elements or character data.

5.3 Page attribute size

A list of two real numbers separated by a comma specifying the Application Media Size as described by the XPS `FixedPage`'s `Width` and `Height` attributes. The first number is the width and the second the height, expressed in 1/96 of an inch.

5.4 Page attribute BleedBox

A list of four real numbers separated by commas, specifying the Application Bleed Size. The order and interpretation of the values is the same as for the XPS `FixedPage`'s `BleedBox` attribute.

5.5 Page attribute ContentBox

A list of four real numbers separated by commas, specifying the Application Content Size. The order and interpretation of the values is the same as for the XPS `FixedPage`'s `ContentBox` attribute.

The page details file is opened and written to after the page start configuration file has been opened, and before any page level configuration PostScript language code has been read. The page details file will be closed after all of its content has been written by the RIP and before any start page configuration PostScript language code is read.

6 Page range

The PrintTicket device can control which pages in a `FixedDocument` are rendered based on the `DocumentPageRanges` PrintTicket parameter. The RIP reads a read-only device parameter to find the index of the next XPS `FixedPage` element to process.

The device parameter is:

```
/NextPageD
```

The value is a 1-based index. There are two special values for the device parameter defined in `interface\xps\printticket.h` that should be returned if applicable:

`XPSPT_PAGES_ALL`
: Informs the RIP that all `FixedPages` in the current `FixedDocument` should be rendered.

`XPSPT_PAGES_NOMORE`
: Informs the RIP that no more `FixedPages` in the current `FixedDocument` should be rendered.

The `NextPage` parameter *must* be implemented by the PrintTicket device. Page numbers range from `1` to `N` where `N` is the number of pages within a `FixedDocument`.

The next page returned *must* be `1-N`, `XPSPT_PAGES_NOMORE` or `XPSPT_PAGES_ALL`.

Returning XPSPT_PAGES_ALL *must* be returned as the first call to this device parameter for any document, at which point no further calls to next page will be made. That is, XPSPT_PAGES_ALL can only be returned on the first call to the NextPage parameter per FixedPage, rather than the whole job.

Returning XPSPT_PAGES_ALL after returning a valid page number will cause an error.

Return XPSPT_PAGES_NOMORE to terminate a list of pages. For example:

```
1,59,3,2,XPSPT_PAGES_NOMORE
```

7 Error handling

There are four possible sources of error which the PrintTicket device needs to handle:

- Incorrect interaction with the PrintTicket device by the RIP.

These are problems that arise with normal device semantics—unknown file name, invalid open flags, and so on. To handle future versions of the interface, the PrintTicket device should ignore any unexpected file names and operations, returning success codes.

- Errors encountered while parsing XML

The error can come from either the XML parser or your own callbacks. The XML parser can catch syntactic errors whilst your callbacks can provide validation errors. In both cases you can set up information to be picked-up and reported by the RIP. The PrintTicket device needs to support the following read-only parameters:

```
/ErrorNo
/ErrorLine
/ErrorColumn
/ErrorMessage
```

The expected parameters are integer for the first three and string for the last. Internally, the core PrintTicket code should check the parameter types and ignore them or fail if they are the wrong type.

The error number should be one of the error numbers currently listed in the include file interface/printticket.h. The error line and column values should be the line and column numbers reported from the XML parser for the start of the element for the current callback. The error message can be anything, but a zero length string will be treated as no message present.

The device error parameters are read after closing the file containing the XML raising the error.

- Errors with configuration PostScript language code

If the RIP encounters an error while consuming configuration PostScript language code there will be no more calls to read data, but there will still be a call to close the file.

- Errors processing the XPS Package

It is possible that the RIP will encounter a problem processing an XPS package (including the processing of PrintTicket parts) and will abort processing it. To allow the PrintTicket device to tidy itself up in this situation, the RIP will write a boolean `true` value to the following device parameter:

```
/AbortJob
```

This parameter will be written to in place of reading from the end scope configuration file. This means for example, that if the RIP aborts processing a package having started to parse an XPS `FixedPage` element, the parameter will be written to three times as it undoes each scope level. This allows you to either; tidy up everything as soon as you detect the parameter being set; or tidy up just what is relevant for the scope.

This parameter will still be written to if the cause of the job being aborted is due to an error raised while parsing the `PageDetails` or PrintTicket XML. It will be written to after the error device parameters have been read. If the RIP encounters an error while consuming the end configuration PostScript language code, the RIP will abort processing the XPS package. However, it will not set the `/AbortJob` device parameter for the PrintTicket scope for which it was consuming configuration PostScript language code, so you should catch the close file call being made before all the configuration PostScript language code has been read and treat that as indicating the job is being aborted.

8 Device implementation

The PrintTicket device supplied is an example implementation that supports the interface described above. Since the PrintTicket device exists as a fully fledged PostScript language device, every interface function needs to be implemented. However, only those functions required to support the interface need to be fully implemented, the remainder can have stubs returning a fixed result—they will not be called when processing an XPS package.

The functions that will need to be implemented are:

```
device_init
device_dismount
device_buffersize
last_error
open_file
read_file
write_file
close_file
abort_file
bytes_file
set_param
start_param
get_param
```

The functions that do not need to be implemented and can return fixed values are:

```
status_device
rename_file
delete_file
seek_file
status_file
start_file_list
next_file
end_file_list
ioctl_call
```

9 Example PrintTicket device interaction

The following is an example sequence of opens, reads/writes, and closes made to the PrintTicket device while processing an XPS package with a single `FixedDocument` with two `FixedPages`, and PrintTickets associated with the`FixedDocumentSequence` and one of the `FixedPages`.

```
JS Open
JS Write   The PrintTicket associated with the FixedDocumentSequence.
JS Read
JS Close

DS Open
DS Read    No write implies no document level PrintTicket in the package.
DS Close
```

```
PS Open
PD Open
PD Write
PD Close
PS Read    No write implies no page level PrintTicket in the package.
PS Close

PE Open
PE Read
PE Close

PS Open
PS Write   The PrintTicket associated with the second FixedPage.
PD Open
PD Write
PD Close
PS Read
PS Close

PE Open
PE Read
PE Close    The second FixedPage's PrintTicket is no longer in scope.

DE Open
DE Read
DE Close

JE Open
JE Read
JE Close
```

10 Document history

Change history		
v1.0	21.03.2013	New Document

Copyright and Trademarks

Harlequin The PrintTicket Device

Version 1.0: March 2013

Part number: Hqn088

Document issue: 100