Jaws overview

Introduction

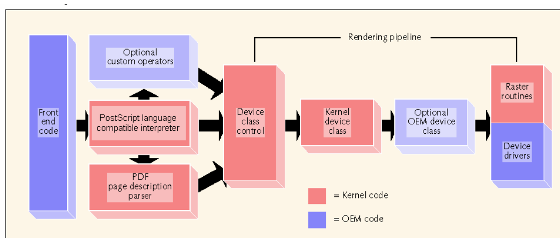
This manual describes the Jaws RIP (raster image processor), a portable LanguageLevel 3 PostScript compatible interpreter that can handle PDF documents up to version 1.7.

Jaws is designed as an SDK (Software Development Kit) that supports an open systems approach to constructing a RIP. This approach means that, aside from the kernel, there is some “glue” code which must be added to produce a complete RIP, although you are free to add as much optional code as you like. Sample code is provided for all the required non-kernel code that illustrates the correct use of most of the kernel-based interfaces and can be used as the starting point for your own code.

It is possible to build a minimal RIP using only the kernel and basic sample code. OEMs are encouraged to add value to the core wherever appropriate. A popular example of such added value is color correction code. The Jaws kernel does not perform any color correction itself (other than that implied by the device-independent colors spaces in the PostScript language itself), but the device class interface allows you to add your own color correction, which can be as simple or as sophisticated as you wish.

While we encourage OEMs to integrate Jaws closely with their own code, you should recognize that Jaws is much larger than a conventional graphics library, and for example, includes its own memory manager which replaces the traditional memory management routines from the C library. This means that you should exercise care when integrating Jaws into larger programs.

The following diagram shows how the major components of Jaws are connected together:



The most important kernel components are:

- The core PostScript compatible interpreter is responsible for overall control. It accepts buffers of input from the front-end code provided by the OEM (based on the sample front-end code provided with Jaws) and parses and executes the PostScript language data.
- The PDF parser is used to render individual page descriptions from PDF files.

There are two different approaches to rendering PDF. The first involves producing a set of PostScript language procedures which will sequentially read an entire PDF file, spooling the objects in the PDF out to temporary files, and then executing these objects as PostScript code. The second approach is to interpret the PDF directly within the interpreter.

The first approach will work on any PostScript interpreter, but may be inefficient and may restrict the flexibility with which the PDF is rendered. Jaws adopts the second approach. This generally offers more efficient processing, but there are a couple of restrictions which should be considered. The first restriction is that PDF is a random-access format.

Jaws may need to seek to random positions in the PDF file whenever objects from the file are used. This implies that the PDF file must be stored on disk (or in RAM, using the RAM file system), and must be complete before you start to render it. Thus, unlike the PostScript language, PDF cannot be rendered a little at a time as it arrives over a network or serial connection.

The second restriction is that (usually) Jaws will only print the page descriptions from a PDF file, and not any annotations on those pages. The one exception to this is that a special operator is provided to render all the annotations on a page. This allows you to efficiently print Acrobat forms data, for example in an on-demand printing application. Jaws renders PDF page descriptions in a similar way to PostScript language forms, and they can be transformed by the current graphics state. This allows you to perform simple imposition with very little extra processing.

Jaws allows you to define custom operators for your own application. Clearly, these operators will not be used inside PostScript page descriptions, unless you are working in a tightly controlled environment. However, custom operators can be useful for passing information from the PostScript language program directly to one or more kernel or OEM device classes.

The remainder of Jaws forms the rendering pipeline, comprising the following components:

The device class control layer administers the various device classes, pushing and popping them from the rendering pipeline as needed. For example, the font cache is implemented as a separate device class, so all the font rendering code in the interpreter must deal with the device class control code to ensure that the cache device class is in place when rendering cached glyphs, and removed from the pipeline when an uncached glyph is rendered.

The phrase *device class* has changed over time. When Jaws was first written, C++ was little more than a curiosity and real classes were generally only used by Lisp programmers, so at the time it seemed a relatively harmless abuse of terminology.

Also “device class” uses the word “class” in a more casual, everyday English sense than is common in software usage today. Whereas a device driver encapsulates the characteristics of a unique output device, such as a particular kind of frame buffer or printer, a device class encapsulates the characteristics of a whole class of output device, such as all windows, or all “frame device” printers, or all “band device” printers, and so on. All these classes of device accept the same set of graphics primitives (**f**
ill, **show**, **image** and so

on), but each implements them differently. Or, where a frame device might fill a path immediately on execution of the **fill** operator, a band device will store it in a display list and fill it later when the bands for the current page are being rendered.

Once this abstraction was made, it was a short step to using device classes for the font and form caches too. Subsequently, OEM-written device classes were also allowed, and now the term “device class” is used to describe any code which sits in the rendering pipeline and uses the device class API to communicate with the rest of Jaws.

There are some cosmetic similarities between a device class and a “real” class. A device class consists of a set of self-contained functions which are called via a publicly defined interface, and which use private data to store state. However, the publicly defined interface is identical for all device classes, comprising as it does the set of primitive graphics operations for the PostScript language, and all calls using this interface go through the device class control routines.

This manual (see Section 10.2) also talks about *device class instances.* The word “instance” is used here to describe a device class after it has been loaded into the rendering pipeline, at which point it also possibly includes some private data. It is possible but unusual for a given device class to have more than one instance active at once. This is normally most useful when the class in question represents the final output to a destination other than bitmaps—in this case you might want to re-use the same code to output geometry to the body of a page description as you do to output it to a character description or a pattern description.

The PostScript compatible interpreter does not contain any graphics code, other than the code which maintains the graphics state, the path construction operators, and so on. All code which actually marks the output page, is implemented entirely by device classes, and all marking operators (**fill**, **show**, **image** and so on) just make calls on the rendering pipeline, via the device class control code.

The body of the rendering pipeline is made up of one or more device classes. If there are more than one class, they are connected in series, and graphics commands (normally) pass through them one after the other. The Jaws kernel contains a number of standard device classes. These are the device classes which are required to implement a working interpreter. The most obvious is the final page output device class, which is responsible for rendering all graphics operations onto the page bitmaps. Other kernel device classes implement the font cache, caching of form and pattern data (as in-memory display lists), execution of previously stored display lists, and banding devices.

In addition to all these, any number of OEM device classes are allowed. Normally, these device classes will implement “added value” features not present in the kernel, such as color correction. In such cases, you must write the necessary device driver code to install your device class in the rendering pipeline and activate it properly. Once it is in the rendering pipeline, of course, the kernel will pass data to it just like any other device class. It is also possible to write your own device class to replace one or more of the standard kernel device classes—this is more difficult, as it involves responding correctly to initialization and other messages sent by the interpreter. One possible use of this technique is to filter the data before passing it on to the original kernel device class—for example applying color correction to a form or pattern before it is recorded in the cache.

Note: You cannot place device class code in a Windows DLL and dynamically load it into Jaws. The calling interface for a device class consists of a small number of well-defined procedures, so it seems that this ought to be possible. However, the total API for a device class must also include any kernel “helper” routines which might be available for the device class to call. This could potentially include any kernel routines, which would mean that every Jaws kernel routine would have to be exported in case a DLL wanted to call it. It was decided that requiring device class code to be statically linked into Jaws was a simpler overall approach.

One corollary of the above is that the documentation for device class helper routines given in this manual is necessarily incomplete. If you are attempting an advanced device class, you should contact support to find out whether there are any kernel routines which would make life easier for you.

The raster routines are used if the final output format is a bitmap, and the last device class in the rendering pipeline is the page output device class. In this case, the page output device class will (under control of the **setpagedevice** operator and the OEM's device driver code) maintain a collection of one or more rasters representing the output page. It will translate graphics commands received from the rendering pipeline into simpler calls which are then made to the raster routines, which mark the output rasters.

The raster routines that are shipped with Jaws handle rasters with pixel depths of 1, 4, 8, 16, 24, 32, 48 and 64-bits per pixel, stored in raw pixel form. It is possible to develop raster routines for other depths and organizations of raster, although this is a major undertaking. The raster API, although generally stable, is not officially documented, and you would need assistance from GGS if you wanted to attempt this.

The OEM's device driver code is responsible for correctly describing the output device capabilities to the kernel **setpagedevice** operator. It also implements any device actions that are required by **showpage** and other related operators. In addition to this, it is responsible for loading any special device classes which are needed by the output device. An extreme example of this is the "xlat" sample code, where the device driver replaces the standard page output device class completely, as the final output produced by this driver is PostScript program code and not bitmaps.

It is possible to build a version of Jaws with several device drivers linked in. In this case you can select just one device driver at run time by using the **setpagedevice** operator. The sample executable delivered with Jaws, for example, is linked with a number of raster based device drivers that allow you to write bitmap files to disk in a variety of different organizations and formats.

The various components of Jaws communicate with each other using fairly simple interfaces. Mostly these are built around function calls and shared data structures. In some cases, such as device classes, structures are allowed to contain function pointers. Nothing in Jaws depends on language capabilities beyond those of "Kernighan and Ritchie" C, except for the use of enumerated data types and void functions, which are almost universally available. Many modern compilers claim to support both C and C++. Some of these compilers incorrectly treat as C keywords some words which are keywords in C++ but not in C—in some cases these clash with identifiers used by the Jaws source code, and in such cases the identifier has been renamed to avoid the clash, so that the operator typedef has become psoperator instead. However, it is not recommended that you try to include Jaws header files into C++ code; any code which interfaces directly with Jaws should preferably be written in C, to minimize the risk of any language-related problems.

Jaws is designed so that all OEM-written code should be as portable as possible across all platforms. So, for example, the sample output device drivers all use the Jaws routines psopen(), psclose() and so on, to write their output to disk, and to remove dependencies on the I/O system calls on different operating systems. There are some extra features (such as the ResourceDecode filter on the Macintosh) which are only available on specific platforms, but, in general, all core programming support should be the same on all platforms. This even extends to the memory manager implemented in Jaws; it is surprisingly often the case that memoryrelated bugs on one platform will be reproducible on another.

Delivering Jaws to your customers

The core set of deliverables for Jaws comprises the following:

- A prebuilt standalone Jaws executable. This provides a benchmark build, which is also used for error reporting. This is important because it provides a known base with which to reproduce errors. Quite often a bug can depend on the exact configuration in which Jaws is being run. Testing a bug with your own copy of the standalone executable before reporting it can ensure that you have correctly reported the configuration, in cases where your own device driver code does not correspond exactly with any of the sample device drivers. Of course, as bugs are fixed, the standalone executable will gradually fall out of date.
- In a build directory containing the Jaws libraries and/or object files, and the necessary make or project files needed to rebuild the standalone executable. You should make a copy of this build directory before starting to make changes to it for your own product. This will enable you to rebuild the Jaws standalone executable, using a current set of kernel libraries, should you need it for error reporting as described above. A complete set of header files, and a minimal front-end implementation are also included.
- The PostScript language support files, containing commonly used PostScript language procedures, and all the common PostScript language resources, such as default halftones, CMap files and so on. Since these are not part of the kernel, it is relatively easy to make changes to these to add convenience procedures and the like. Any substantial changes should probably be implemented as ProcSet resources, once again for the purpose of keeping your base version of Jaws close to the normal standalone.
- A set of 35 standard fonts. They are, of course, licensed only for use with Jaws. The 100+ extra fonts which ship with PostScript LL3 products are not supplied with Jaws. Normally, this is not a problem as you can produce a PPD (PostScript Printer Description) file for your product which does not declare these extra fonts, and the printer driver will then embed the fonts in documents as it prints them. If you need to ship the extra fonts, there are a number of potential sources for them, which GGS can suggest to you, according to your selection criteria, such as quality at low

resolution, availability of matching screen fonts, and so on.

- Sample device driver code. This comprises a set of device drivers, all of which write their output to files on disk. The differences between them are intended to show how you would implement devices with different capabilities, such as frame-based devices or banding devices, monochrome or color devices, separated output, devices with multiple paper sources and so on. In many cases, you can just use these sample device drivers unmodified, for example if you always spool output to disk before printing. In other cases, you will want to use these device drivers as the starting point for your own code.
- Sample device class code. This provides some illustrative examples of how to use the device class interface, and is intended to accompany the device class documentation in the manual. For more information see Chapter 10, "Device classes".
- Example userfs-based Jaws file systems. The userfs file system allows OEMs to create their own file systems for Jaws.
- Colourdv shows how to add color correction to Jaws using 3rd party CMMs (ColorSync, LittleCMS, or WCS).
- pdfcrypt shows how to add support for encrypted PDFs using 3rd party encryption code (OpenSSL or WinCrypt).
- PDF Access shows how Jaws can be used to gain access at the object level to the content of a PDF. For example, this could be used to extract the PJTF from a PDF in order to configure Jaws prior to printing the job.

Jaws support

Support for the Jaws RIP is available from Jaws Support at: jaws-support@globalgraphics.com.

To submit a request for technical support, a form is available at the following address:

https://www.globalgraphics.com/support/JOEMsupp.nsf/IssueWeb

Enter your Jaws user name and password to access the form. Evaluation customers may use the login details given below:

username: `jaws-evalu`
password: `teMPReF43`

Note: Contact Jaws Support if you do not have your login details.

Testing your code with the Jaws test-harness

The Jaws test-harness allows you to render PostScript language and PDF files in a simple and reproducible environment. You should always test any problem file(s) with the test-harness before requesting technical assistance from Jaws Support.

For instructions on building the test harness see the HTML documentation supplied with the Jaws SDK.

When testing the issue in the test harness, please make sure you are using the latest version of Jaws. Since Jaws is often updated, it is very possible that an issue you are experiencing has already been discovered and fixed in the latest update to Jaws. You can access the latest files for Jaws by logging on to our FTP site located at ftp://support.globalgraphics.com.

Jaws OEMs receive "Welcome to Jaws Support" e-mail which includes your FTP login details. If you require this information again, contact Jaws Support at the e-mail address listed above.

The latest files for Jaws will be located in the directory:

```
/Users/<OEM user-name>
```

You will need the latest libraries, include, lib and resource files. Once you have updated your Jaws files, ensure that all previous intermediate object created during previous compilations are removed prior to rebuilding and completely rebuild Jaws from scratch. If you are unable to reproduce the problem on the stand-alone test-harness then the simplest solution is to add any custom code that you may be using to the stand-alone bit by bit until it goes wrong too. This may help you solve the problem yourself. If not, then you can send the code to us and we will then be able to look at the problem. However, please do not send code without contacting us first. If you are asked to send code, do not modify one of the standard device drivers. If you need to send a modified device driver, please give it a different name. If you cannot do any of this then we will still try and help you, but it will be very difficult to do so and will significantly increase the time taken to address the problem.

Guidelines for submitting a Jaws support request

To ensure an efficient and speedy resolution to your Jaws issue, the following guidelines should always be followed when submitting a support request to Jaws Support:

1. Always test the problem file(s) with the Jaws RIP test-harness before requesting technical assistance. Doing this will ensure the problem is with Jaws and not with your own code. See *Testing your code with the Jaws test-harness* for details on how to use the Jaws test-harness.

2. To submit a request for technical assistance, use the support form at: https://www.globalgraphics.com/support/JOEMsupp.nsf/IssueWeb
All subsequent contacts on this topic should be made by sending mail to jaws-support@globalgraphics.com. The subject line of the e-mail must remain unchanged as this includes properly formatted tracking information.

3. Include the following system/RIP details when reporting an issue:

- Operating system details. For example, Windows 2008 R2.
- Before reporting a problem, please confirm that you are using one of our supported development environments.
- Kernel build number. This can be obtained by running:

currentsystemparams /KernelBuild get ==

Be aware that you may need to add a flush to the end of the PostScript code line. You can either put this in a file and run that, or type executive at the Jaws Window (Please note this will NOT work for Unix systems) and then type in the line of PostScript language code above.

- Total system RAM and details of how much memory is allocated to the RIP.
- If you receive an error message or warning, please include the full and complete text of the message.
- RIP configuration (**userparams**, **/OutputDevice**, and so on).
- If possible, provide details of any previous Jaws build where the problem did not occur.

4. Be as specific as possible when describing the problem you are seeing. Descriptions like “corrupted image” to describe a problem with one of 30 images on a 36” square output are not specific enough. A low resolution screen shot often helps too.

5. Give the complete and actual /OutputDevice string required to reproduce the problem. This is extremely important as many bugs manifest only at certain resolutions or color depth, or only under certain memory configurations. We cannot begin investigating your report until we have this information. Feel free to include your init.ps file with your example file.

6. Give any other information required to reproduce the problem. This should be included in the “Extra Configuration Details”. Information that should be entered here includes any **userparams**, **pagedevice** keys, **systemparams**, and so on, that you have set.

7. If a specific test file is needed to reproduce the issue you can upload it to our FTP site.

Please ensure that all test files are reduced to a minimum, while still reproducing the issue. This means removing extraneous images, text and objects that are irrelevant to the actual issue. We also recommend that you compress all files before uploading.

8. Unless page size or image resolution is fundamental to the issue, you should report problems on small page sizes, using a low resolution. Obviously, if the problem only occurs at 5000 dpi on A0 media, report this as the problem. But generally it is quicker to reproduce the problem if the output/resolution is small.

All questions regarding these procedures should be addressed to Jaws Support at: jaws-support@globalgraphics.com

General considerations

Normally, you should treat bug reporting for Jaws in the same way that you treat bug reporting for any other compiler or interpreter. Remember that the more effort you put into isolating and characterizing a bug, the sooner we will be able to supply you with a fix.

Try to make sure that each file exhibits only one problem. Try to submit the smallest file that exhibits the problem.

Try to reproduce the problem in such a way that we can easily reproduce it here. (If you habitually render to a 100 MB+ continuous tone frame buffer, we are going to have problems rendering with the same configuration as you, and we will also have problems trying to view the output.)

If the problem is one of incorrect output (as opposed to a program crash), try to describe as precisely as possible what you see. If the problem is that no output is produced, try to describe as precisely as possible what happens (do you get a segmentation fault, a bus error, or does Jaws hang?). Phrases like “it falls over” can be ambiguous.

Do not try to speculate about the possible cause. If you do have some helpful suggestions of which you are not sure, make sure that you point out they are speculation in your bug report.

Describe the exact configuration under which the problem occurs. Remember that we do not know what your front-end software looks like.

For platforms on which Jaws allocates a fixed amount of memory when it starts up, we need to know how much memory you are giving Jaws. On all platforms, it is vitally important that you report the **OutputDevice** string you use to render the job. (There are too many possibilities for us to try them at random.) List the platform(s) on which you have reproduced the bug, and any on which you know it does not appear. If you are using non-standard halftones, color rendering and so on, this may also be a factor affecting the way in which we can reproduce the problem.

All these details are requested on the Bug Report Form and it is essential that it is correctly and accurately completed.