Directness and Simplicity

The JSON output contains the value of every object in the file, but it also contains some processed data. This is analogous to how qpdf's library interface works. The processed data is similar to the helper functions in that it allows you to look at certain aspects of the PDF file without having to understand all the nuances of the PDF specification, while the raw objects allow you to mine the PDF for anything that the higher-level interfaces are lacking.

6.3. Limitations of JSON Representation

There are a few limitations to be aware of with the JSON structure:

- Strings, names, and indirect object references in the original PDF file are all converted to strings in the JSON representation. In the case of a "normal" PDF file, you can tell the difference because a name starts with a slash (`/`), and an indirect object reference looks like `n n R`, but if there were to be a string that looked like a name or indirect object reference, there would be no way to tell this from the JSON output. Note that there are certain cases where you know for sure what something is, such as knowing that dictionary keys in objects are always names and that certain things in the higher-level computed data are known to contain indirect object references.

- The JSON format doesn't support binary data very well. Mostly the details are not important, but they are presented here for information. When qpdf outputs a string in the JSON representation, it converts the string to UTF-8, assuming usual PDF string semantics. Specifically, if the original string is UTF-16, it is converted to UTF-8. Otherwise, it is assumed to have PDF doc encoding, and is converted to UTF-8 with that assumption. This causes strange things to happen to binary strings. For example, if you had the binary string `<038051>`, this would be output to the JSON as `\u0003•Q` because `03` is not a printable character and `80` is the bullet character in PDF doc encoding and is mapped to the Unicode value `2022`. Since `51` is `Q`, it is output as is. If you wanted to convert back from here to a binary string, would have to recognize Unicode values whose code points are higher than `0xFF` and map those back to their corresponding PDF doc encoding characters. There is no way to tell the difference between a Unicode string that was originally encoded as UTF-16 or one that was converted from PDF doc encoding. In other words, it's best if you don't try to use the JSON format to extract binary strings from the PDF file, but if you really had to, it could be done. Note that qpdf's **--show-object** option does not have this limitation and will reveal the string as encoded in the original file.

6.4. JSON: Special Considerations

For the most part, the built-in JSON help tells you everything you need to know about the JSON format, but there are a few non-obvious things to be aware of:

- While qpdf guarantees that keys present in the help will be present in the output, those fields may be null or empty if the information is not known or absent in the file. Also, if you specify **--json-keys**, the keys that are not listed will be excluded entirely except for those that **--json-help** says are always present.

- In a few places, there are keys with names containing `pageposfrom1`. The values of these keys are null or an integer. If an integer, they point to a page index within the file numbering from 1. Note that JSON indexes from 0, and you would also use 0-based indexing using the API. However, 1-based indexing is easier in this case because the command-line syntax for specifying page ranges is 1-based. If you were going to write a program that looked through the JSON for information about specific pages and then use the command-line to extract those pages, 1-based indexing is easier. Besides, it's more convenient to subtract 1 from a program in a real programming language than it is to add 1 from shell code.

- The image information included in the `page` section of the JSON output includes the key "`filterable`". Note that the value of this field may depend on the **--decode-level** that you invoke qpdf with. The JSON output includes a top-level key "`parameters`" that indicates the decode level used for computing whether a stream was filterable.