8.4. Writing Linearized Files

We will create files with only primary hint streams. We will never write overflow hint streams. (As of PDF version 1.4, Acrobat doesn't either, and they are never necessary.) The hint streams contain offset information to objects that point to where they would be if the hint stream were not present. This means that we have to calculate all object positions before we can generate and write the hint table. This means that we have to generate the file in two passes. To make this reliable, ***QPDFWriter*** in linearization mode invokes exactly the same code twice to write the file to a pipeline.

In the first pass, the target pipeline is a count pipeline chained to a discard pipeline. The count pipeline simply passes its data through to the next pipeline in the chain but can return the number of bytes passed through it at any intermediate point. The discard pipeline is an end of line pipeline that just throws its data away. The hint stream is not written and dummy values with adequate padding are stored in the first cross reference table, linearization parameter dictionary, and /Prev key of the first trailer dictionary. All the offset, length, object renumbering information, and anything else we need for the second pass is stored.

At the end of the first pass, this information is passed to the ***QPDF*** class which constructs a compressed hint stream in a memory buffer and returns it. ***QPDFWriter*** uses this information to write a complete hint stream object into a memory buffer. At this point, the length of the hint stream is known.

In the second pass, the end of the pipeline chain is a regular file instead of a discard pipeline, and we have known values for all the offsets and lengths that we didn't have in the first pass. We have to adjust offsets that appear after the start of the hint stream by the length of the hint stream, which is known. Anything that is of variable length is padded, with the padding code surrounding any writing code that differs in the two passes. This ensures that changes to the way things are represented never results in offsets that were gathered during the first pass becoming incorrect for the second pass.

Using this strategy, we can write linearized files to a non-seekable output stream with only a single pass to disk or wherever the output is going.

8.5. Calculating Linearization Data

Once a file is optimized, we have information about which objects access which other objects. We can then process these tables to decide which part (as described in “Linearized PDF Document Structure” in the PDF specification) each object is contained within. This tells us the exact order in which objects are written. The ***QPDFWriter*** class asks for this information and enqueues objects for writing in the proper order. It also turns on a check that causes an exception to be thrown if an object is encountered that has not already been queued. (This could happen only if there were a bug in the traversal code used to calculate the linearization data.)

8.6. Known Issues with Linearization

There are a handful of known issues with this linearization code. These issues do not appear to impact the behavior of linearized files which still work as intended: it is possible for a web browser to begin to display them before they are fully downloaded. In fact, it seems that various other programs that create linearized files have many of these same issues. These items make reference to terminology used in the linearization appendix of the PDF specification.

- Thread Dictionary information keys appear in part 4 with the rest of Threads instead of in part 9. Objects in part 9 are not grouped together functionally.
- We are not calculating numerators for shared object positions within content streams or interleaving them within content streams.
- We generate only page offset, shared object, and outline hint tables. It would be relatively easy to add some additional tables. We gather most of the information needed to create thumbnail hint tables. There are comments in the code about this.