Chapter 8. Linearization

This chapter describes how ***QPDF*** and ***QPDFWriter*** implement creation and processing of linearized PDFS.

8.1. Basic Strategy for Linearization

To avoid the incestuous problem of having the qpdf library validate its own linearized files, we have a special linearized file checking mode which can be invoked via **qpdf --check-linearization** (or **qpdf --check**). This mode reads the linearization parameter dictionary and the hint streams and validates that object ordering, parameters, and hint stream contents are correct. The validation code was first tested against linearized files created by external tools (Acrobat and pdlin) and then used to validate files created by ***QPDFWriter*** itself.

8.2. Preparing For Linearization

Before creating a linearized PDF file from any other PDF file, the PDF file must be altered such that all page attributes are propagated down to the page level (and not inherited from parents in the `/Pages` tree). We also have to know which objects refer to which other objects, being concerned with page boundaries and a few other cases. We refer to this part of preparing the PDF file as *optimization*, discussed in Section 8.3, “Optimization”, page 37. Note the, in this context, the term *optimization* is a qpdf term, and the term *linearization* is a term from the PDF specification. Do not be confused by the fact that many applications refer to linearization as optimization or web optimization.

When creating linearized PDF files from optimized PDF files, there are really only a few issues that need to be dealt with:

- Creation of hints tables
- Placing objects in the correct order
- Filling in offsets and byte sizes

8.3. Optimization

In order to perform various operations such as linearization and splitting files into pages, it is necessary to know which objects are referenced by which pages, page thumbnails, and root and trailer dictionary keys. It is also necessary to ensure that all page-level attributes appear directly at the page level and are not inherited from parents in the pages tree.

We refer to the process of enforcing these constraints as *optimization*. As mentioned above, note that some applications refer to linearization as optimization. Although this optimization was initially motivated by the need to create linearized files, we are using these terms separately.

PDF file optimization is implemented in the *QPDF_optimization.cc* source file. That file is richly commented and serves as the primary reference for the optimization process.

After optimization has been completed, the private member variables *obj_user_to_objects* and *object_to_obj_users* in ***QPDF*** have been populated. Any object that has more than one value in the *object_to_obj_users* table is shared. Any object that has exactly one value in the *object_to_obj_users* table is private. To find all the private objects in a page or a trailer or root dictionary key, one merely has make this determination for each element in the *obj_user_to_objects* table for the given page or key.

Note that pages and thumbnails have different object user types, so the above test on a page will not include objects referenced by the page's thumbnail dictionary and nothing else.