Chapter 7. Design and Library Notes

7.1. Introduction

This section was written prior to the implementation of the qpdf package and was subsequently modified to reflect the implementation. In some cases, for purposes of explanation, it may differ slightly from the actual implementation. As always, the source code and test suite are authoritative. Even if there are some errors, this document should serve as a road map to understanding how this code works.

In general, one should adhere strictly to a specification when writing but be liberal in reading. This way, the product of our software will be accepted by the widest range of other programs, and we will accept the widest range of input files. This library attempts to conform to that philosophy whenever possible but also aims to provide strict checking for people who want to validate PDF files. If you don't want to see warnings and are trying to write something that is tolerant, you can call `setSuppressWarnings(true)`. If you want to fail on the first error, you can call `setAttemptRecovery(false)`. The default behavior is to generating warnings for recoverable problems. Note that recovery will not always produce the desired results even if it is able to get through the file. Unlike most other PDF files that produce generic warnings such as “This file is damaged,”, qpdf generally issues a detailed error message that would be most useful to a PDF developer. This is by design as there seems to be a shortage of PDF validation tools out there. This was, in fact, one of the major motivations behind the initial creation of qpdf.

7.2. Design Goals

The QPDF package includes support for reading and rewriting PDF files. It aims to hide from the user details involving object locations, modified (appended) PDF files, the directness/indirectness of objects, and stream filters including encryption. It does not aim to hide knowledge of the object hierarchy or content stream contents. Put another way, a user of the qpdf library is expected to have knowledge about how PDF files work, but is not expected to have to keep track of bookkeeping details such as file positions.

A user of the library never has to care whether an object is direct or indirect, though it is possible to determine whether an object is direct or not if this information is needed. All access to objects deals with this transparently. All memory management details are also handled by the library.

The ***PointerHolder*** object is used internally by the library to deal with memory management. This is basically a smart pointer object very similar in spirit to C++-11's ***std::shared_ptr*** object, but predating it by several years. This library also makes use of a technique for giving fine-grained access to methods in one class to other classes by using public subclasses with friends and only private members that in turn call private methods of the containing class. See ***QPDFObjectHandle::Factory*** as an example.

The top-level qpdf class is ***QPDF***. A ***QPDF*** object represents a PDF file. The library provides methods for both accessing and mutating PDF files.

The primary class for interacting with PDF objects is ***QPDFObjectHandle***. Instances of this class can be passed around by value, copied, stored in containers, etc. with very low overhead. Instances of ***QPDFObjectHandle*** created by reading from a file will always contain a reference back to the ***QPDF*** object from which they were created. A ***QPDFObjectHandle*** may be direct or indirect. If indirect, the ***QPDFObject*** the ***PointerHolder*** initially points to is a null pointer. In this case, the first attempt to access the underlying ***QPDFObject*** will result in the ***QPDFObject*** being resolved via a call to the referenced ***QPDF*** instance. This makes it essentially impossible to make coding errors in which certain things will work for some PDF files and not for others based on which objects are direct and which objects are indirect.

Instances of ***QPDFObjectHandle*** can be directly created and modified using static factory methods in the ***QPDFObjectHandle*** class. There are factory methods for each type of object as well as a convenience method *QPDFObjec-*