itable resources from the `/Pages` tree down to individual pages and manipulation of the `/Pages` tree itself. For details, see *addPage* and surrounding methods in *QPDF.hh*.

7.9. Reserving Object Numbers

Version 3.0 of qpdf introduced the concept of reserved objects. These are seldom needed for ordinary operations, but there are cases in which you may want to add a series of indirect objects with references to each other to a ***QPDF*** object. This causes a problem because you can't determine the object ID that a new indirect object will have until you add it to the ***QPDF*** object with *QPDF::makeIndirectObject*. The only way to add two mutually referential objects to a ***QPDF*** object prior to version 3.0 would be to add the new objects first and then make them refer to each other after adding them. Now it is possible to create a *reserved object* using *QPDFObjectHandle::newReserved*. This is an indirect object that stays "unresolved" even if it is queried for its type. So now, if you want to create a set of mutually referential objects, you can create reservations for each one of them and use those reservations to construct the references. When finished, you can call *QPDF::replaceReserved* to replace the reserved objects with the real ones. This functionality will never be needed by most applications, but it is used internally by QPDF when copying objects from other PDF files, as discussed in Section 7.10, "Copying Objects From Other PDF Files", page 34. For an example of how to use reserved objects, search for *newReserved* in *test_driver.cc* in qpdf's sources.

7.10. Copying Objects From Other PDF Files

Version 3.0 of qpdf introduced the ability to copy objects into a ***QPDF*** object from a different ***QPDF*** object, which we refer to as *foreign objects*. This allows arbitrary merging of PDF files. The "from" ***QPDF*** object must remain valid after the copy as discussed in the note below. The **qpdf** command-line tool provides limited support for basic page selection, including merging in pages from other files, but the library's API makes it possible to implement arbitrarily complex merging operations. The main method for copying foreign objects is *QPDF::copyForeignObject*. This takes an indirect object from another ***QPDF*** and copies it recursively into this object while preserving all object structure, including circular references. This means you can add a direct object that you create from scratch to a ***QPDF*** object with *QPDF::makeIndirectObject*, and you can add an indirect object from another file with *QPDF::copyForeignObject*. The fact that *QPDF::makeIndirectObject* does not automatically detect a foreign object and copy it is an explicit design decision. Copying a foreign object seems like a sufficiently significant thing to do that it should be done explicitly.

The other way to copy foreign objects is by passing a page from one ***QPDF*** to another by calling *QPDF::addPage*. In contrast to *QPDF::makeIndirectObject*, this method automatically distinguishes between indirect objects in the current file, foreign objects, and direct objects.

Please note: when you copy objects from one ***QPDF*** to another, the source ***QPDF*** object must remain valid until you have finished with the destination object. This is because the original object is still used to retrieve any referenced stream data from the copied object.

7.11. Writing PDF Files

The qpdf library supports file writing of ***QPDF*** objects to PDF files through the ***QPDFWriter*** class. The ***QPDFWriter*** class has two writing modes: one for non-linearized files, and one for linearized files. See Chapter 8, *Linearization*, page 37 for a description of linearization is implemented. This section describes how we write non-linearized files including the creation of QDF files (see Chapter 4, *QDF Mode*, page 22.

This outline was written prior to implementation and is not exactly accurate, but it provides a correct "notional" idea of how writing works. Look at the code in ***QPDFWriter*** for exact details.

- Initialize state:
  - next object number = 1