tHandle::parse that creates an object from a string representation of the object. Existing instances of ***QPDFObjectHandle*** can also be modified in several ways. See comments in *QPDFObjectHandle.hh* for details.

An instance of ***QPDF*** is constructed by using the class's default constructor. If desired, the ***QPDF*** object may be configured with various methods that change its default behavior. Then the *QPDF::processFile()* method is passed the name of a PDF file, which permanently associates the file with that QPDF object. A password may also be given for access to password-protected files. QPDF does not enforce encryption parameters and will treat user and owner passwords equivalently. Either password may be used to access an encrypted file. [1] ***QPDF*** will allow recovery of a user password given an owner password. The input PDF file must be seekable. (Output files written by ***QPDFWriter*** need not be seekable, even when creating linearized files.) During construction, ***QPDF*** validates the PDF file's header, and then reads the cross reference tables and trailer dictionaries. The ***QPDF*** class keeps only the first trailer dictionary though it does read all of them so it can check the `/Prev` key. ***QPDF*** class users may request the root object and the trailer dictionary specifically. The cross reference table is kept private. Objects may then be requested by number of by walking the object tree.

When a PDF file has a cross-reference stream instead of a cross-reference table and trailer, requesting the document's trailer dictionary returns the stream dictionary from the cross-reference stream instead.

There are some convenience routines for very common operations such as walking the page tree and returning a vector of all page objects. For full details, please see the header files *QPDF.hh* and *QPDFObjectHandle.hh*. There are also some additional helper classes that provide higher level API functions for certain document constructions. These are discussed in Section 7.3, “Helper Classes”, page 29.

7.3. Helper Classes

QPDF version 8.1 introduced the concept of helper classes. Helper classes are intended to contain higher level APIs that allow developers to work with certain document constructs at an abstraction level above that of ***QPDFObjectHandle*** while staying true to qpdf's philosophy of not hiding document structure from the developer. As with qpdf in general, the goal is take away some of the more tedious bookkeeping aspects of working with PDF files, not to remove the need for the developer to understand how the PDF construction in question works. The driving factor behind the creation of helper classes was to allow the evolution of higher level interfaces in qpdf without polluting the interfaces of the main top-level classes ***QPDF*** and ***QPDFObjectHandle***.

There are two kinds of helper classes: *document* helpers and *object* helpers. Document helpers are constructed with a reference to a ***QPDF*** object and provide methods for working with structures that are at the document level. Object helpers are constructed with an instance of a ***QPDFObjectHandle*** and provide methods for working with specific types of objects.

Examples of document helpers include ***QPDFPageDocumentHelper***, which contains methods for operating on the document's page trees, such as enumerating all pages of a document and adding and removing pages; and ***QPDFAcroFormDocumentHelper***, which contains document-level methods related to interactive forms, such as enumerating form fields and creating mappings between form fields and annotations.

Examples of object helpers include ***QPDFPageObjectHelper*** for performing operations on pages such as page rotation and some operations on content streams, ***QPDFFormFieldObjectHelper*** for performing operations related to interactive form fields, and ***QPDFAnnotationObjectHelper*** for working with annotations.

It is always possible to retrieve the underlying ***QPDF*** reference from a document helper and the underlying ***QPDFObjectHandle*** reference from an object helper. Helpers are designed to be helpers, not wrappers. The intention is that, in general, it is safe to freely intermix operations that use helpers with operations that use the underlying objects. Document and object helpers do not attempt to provide a complete interface for working with the things they are

[1] As pointed out earlier, the intention is not for qpdf to be used to bypass security on files. but as any open source PDF consumer may be easily modified to bypass basic PDF document security, and qpdf offers may transformations that can do this as well, there seems to be little point in the added complexity of conditionally enforcing document security.