options are effectively operating on the recovered file. Combining **--check** with other options in this way can be useful for manually recovering severely damaged files.

The **--raw-stream-data** and **--filtered-stream-data** options are ignored unless **--show-object** is given. Either of these options will cause the stream data to be written to standard output. In order to avoid commingling of stream data with other output, it is recommend that these objects not be combined with other test/inspection options.

If **--filtered-stream-data** is given and **--normalize-content=y** is also given, qpdf will attempt to normalize the stream data as if it is a page content stream. This attempt will be made even if it is not a page content stream, in which case it will produce unusable results.

3.10. Unicode Passwords

At the library API level, all methods that perform encryption and decryption interpret passwords as strings of bytes. It is up to the caller to ensure that they are appropriately encoded. Starting with qpdf version 8.4.0, qpdf will attempt to make this easier for you when interact with qpdf via its command line interface. The PDF specification requires passwords used to encrypt files with 40-bit or 128-bit encryption to be encoded with PDF Doc encoding. This encoding is a single-byte encoding that supports ISO-Latin-1 and a handful of other commonly used characters. It has a large overlap with Windows ANSI but is not exactly the same. There is generally not a way to provide PDF Doc encoded strings on the command line. As such, qpdf versions prior to 8.4.0 would often create PDF files that couldn't be opened with other software when given a password with non-ASCII characters to encrypt a file with 40-bit or 128-bit encryption. Starting with qpdf 8.4.0, qpdf recognizes the encoding of the parameter and transcodes it as needed. The rest of this section provides the details about exactly how qpdf behaves. Most users will not need to know this information, but it might be useful if you have been working around qpdf's old behavior or if you are using qpdf to generate encrypted files for testing other PDF software.

A note about Windows: when qpdf builds, it attempts to determine what it has to do to use *wmain* instead of *main* on Windows. The *wmain* function is an alternative entry point that receives all arguments as UTF-16-encoded strings. When qpdf starts up this way, it converts all the strings to UTF-8 encoding and then invokes the regular main. This means that, as far as qpdf is concerned, it receives its command-line arguments with UTF-8 encoding, just as it would in any modern Linux or UNIX environment.

If a file is being encrypted with 40-bit or 128-bit encryption and the supplied password is not a valid UTF-8 string, qpdf will fall back to the behavior of interpreting the password as a string of bytes. If you have old scripts that encrypt files by passing the output of **iconv** to qpdf, you no longer need to do that, but if you do, qpdf should still work. The only exception would be for the extremely unlikely case of a password that is encoded with a single-byte encoding but also happens to be valid UTF-8. Such a password would contain strings of even numbers of characters that alternate between accented letters and symbols. In the extremely unlikely event that you are intentionally using such passwords and qpdf is thwarting you by interpreting them as UTF-8, you can use **--password-mode=bytes** to suppress qpdf's automatic behavior.

The **--password-mode** option, as described earlier in this chapter, can be used to change qpdf's interpretation of supplied passwords. There are very few reasons to use this option. One would be the unlikely case described in the previous paragraph in which the supplied password happens to be valid UTF-8 but isn't supposed to be UTF-8. Your best bet would be just to provide the password as a valid UTF-8 string, but you could also use **--password-mode=bytes**. Another reason to use **--password-mode=bytes** would be to intentionally generate PDF files encrypted with passwords that are not properly encoded. The qpdf test suite does this to generate invalid files for the purpose of testing its password recovery capability. If you were trying to create intentionally incorrect files for a similar purposes, the **bytes** password mode can enable you to do this.

When qpdf attempts to decrypt a file with a password that contains non-ASCII characters, it will generate a list of alternative passwords by attempting to interpret the password as each of a handful of different coding systems and then transcode them to the required format. This helps to compensate for the supplied password being given in the wrong coding system, such as would happen if you used the **iconv** workaround that was previously needed. It also generates