combination of the above. As a last resort, a compiler-specific `#pragma` may be used to suppress a warning that we don't want to fix. Examples may include suppressing warnings about the use of old-style casts in code that is shared between C and C++ code.

The casting policy explicitly prohibits casting between integer sizes for no purpose other than to quiet a compiler warning when there is no reasonable chance of a problem resulting. The reason for this exclusion is that the practice of adding these additional casts precludes future use of additional compiler warnings as a tool for making future improvements to this aspect of the code, and it also damages the readability of the code.

There are a few significant areas where casting is common in the qpdf sources or where casting would be required to quiet higher levels of compiler warnings but is omitted at present:

- `char` vs. `unsigned char`. For historical reasons, there are a lot of places in qpdf's internals that deal with `unsigned char`, which means that a lot of casting is required to interoperate with standard library calls and `std::string`. In retrospect, qpdf should have probably used regular (signed) `char` and `char*` everywhere and just cast to `unsigned char` when needed, but it's too late to make that change now. There are *reinterpret_cast* calls to go between `char*` and `unsigned char*`, and there are *static_cast* calls to go between `char` and `unsigned char`. These should always be safe.

- Non-const `unsigned char*` used in the `Pipeline` interface. The pipeline interface has a *write* call that uses `unsigned char*` without a `const` qualifier. The main reason for this is to support pipelines that make calls to third-party libraries, such as zlib, that don't include `const` in their interfaces. Unfortunately, there are many places in the code where it is desirable to have `const char*` with pipelines. None of the pipeline implementations in qpdf currently modify the data passed to write, and doing so would be counter to the intent of `Pipeline`, but there is nothing in the code to prevent this from being done. There are places in the code where *const_cast* is used to remove the const-ness of pointers going into `Pipeline`s. This could theoretically be unsafe, but there is adequate testing to assert that it is safe and will remain safe in qpdf's code.

- `size_t` vs. `qpdf_offset_t`. This is pretty much unavoidable since sizes are unsigned types and offsets are signed types. Whenever it is necessary to seek by an amount given by a `size_t`, it becomes necessary to mix and match between `size_t` and `qpdf_offset_t`. Additionally, qpdf sometimes treats memory buffers like files (as with `BufferInputSource`, and those seek interfaces have to be consistent with file-based input sources. Neither gcc nor MSVC give warnings for this case by default, but both have warning flags that can enable this. (MSVC: **/W14267** or **/W3**, which also enables some additional warnings that we ignore; gcc: **-Wconversion -Wsign-conversion**). This could matter for files whose sizes are larger than 2^{63} bytes, but it is reasonable to expect that a world where such files are common would also have larger `size_t` and `qpdf_offset_t` types in it. On most 64-bit systems at the time of this writing (the release of version 4.1.0 of qpdf), both `size_t` and `qpdf_offset_t` are 64-bit integer types, while on many current 32-bit systems, `size_t` is a 32-bit type while `qpdf_offset_t` is a 64-bit type. I am not aware of any cases where 32-bit systems that have `size_t` smaller than `qpdf_offset_t` could run into problems. Although I can't conclusively rule out the possibility of such problems existing, I suspect any cases would be pretty contrived. In the event that someone should produce a file that qpdf can't handle because of what is suspected to be issues involving the handling of `size_t` vs. `qpdf_offset_t` (such files may behave properly on 64-bit systems but not on 32-bit systems because they have very large embedded files or streams, for example), the above mentioned warning flags could be enabled and all those implicit conversions could be carefully scrutinized. (I have already gone through that exercise once in adding support for files larger than 4 GB in size.) I continue to be committed to supporting large files on 32-bit systems, but I would not go to any lengths to support corner cases involving large embedded files or large streams that work on 64-bit systems but not on 32-bit systems because of `size_t` being too small. It is reasonable to assume that anyone working with such files would be using a 64-bit system anyway since many 32-bit applications would have similar difficulties.

- `size_t` vs. `int` or `long`. There are some cases where `size_t` and `int` or `long` or `size_t` and `unsigned int` or `unsigned long` are used interchangeably. These cases occur when working with very small amounts of memory, such as with the bit readers (where we're working with just a few bytes at a time), some cases of *strlen*, and a few other cases. I have scrutinized all of these cases and determined them to be safe, but there is no mechanism in the code to ensure that new unsafe conversions between `int` and `size_t` aren't introduced short of good testing