Why can I not access files during unit tests?

Unless you are testing a file utility class, if you access files during your test, you exceeding the scope of the *unit* that you should be testing.

That statement sums up the principle behind this rule. However, there are more pragmatic reasons to consider as well.

Clarity / Evident Data

Often the motivation for violating this principle is to read test data. Separating test data from the code of your test class makes that test significantly harder to understand and debug.

In his TDD book, Kent Beck alludes to this practice as *Evident Data*. Show the reader of your test the intent of the code being tested by using clear and understandable data as parameters to your code and expected results in your assert statements. Once test data moves to a separate file, especially in binary formats, the maintenance cost and usefulness of that test becomes worrisome.

Speed Of Execution

Reading files is much slower than executing code. Most

Ease of Setup and Brittleness Issues

Unit tests that depend on files being in the right place in order to pass can no longer be executed simply with a test runner and proper classpath. They must be executed in the context of a build which ensures that the files used are in the right place to be read by the test. This raises maintenance concerns, since changes to the test files would cause your unit tests to break in possibly hard-to-understand ways.

Design Considerations

Most of the time what you really care about is streams. Your design should be flexible enough to deal with other sources than files. By accounting for this you will also get a more flexible design.

Recommendations

- Design all classes that have a legitimate need to read files be coupled to the rest of your codebase via interfaces.
- In some cases, consider using URLs instead of File classes. URLs have handlers which can be mocked out quite easily.
- Create a separate, non-Ashcroft integration test suite with tests to cover implementations that read files.