BackwardsCompatibility

See Java_1.5_Migration for information on our move to Java 1.5

Core Lucene Functionality

Backwards Compatibility in Lucene is all about version numbers.

APIs and Compilation

Minor versions should always have back-compatiblity for supported APIs (i.e. public and protected). That's to say, any code developed against **X.0** should continue to run without alteration and without recompilation (ie, simply drop in the new JAR) against all **X.N** releases, as long as only supported APIs are used. Special exception back compat breaks will be documented in the "Changes in backwards compatibility policy" section of CHANGES. For major releases, there should also be a migration guide to assist in migrating application code across API breaks.

A major release may introduce incompatible API and runtime changes. Within minor versions, the transition strategy is to introduce new APIs in release **X.N** , deprecating old APIs, then remove all deprecated APIs in release **X+1.0**. In some cases, especially with large refactorings across major versions, there will be no deprecation path and you will need to read CHANGES and a migration guide to migrate your code.

Package-private APIs (e.g. methods without any modifier such as public or protected) and those marked as experimental or internal are not supported and thus exempt from these back-compatibility requirements. They can change without warning or intermediate deprecation.

File Formats

File formats are back-compatible between major versions. Version **X.N** should be able to read indexes generated by any version after and including version **X-1.0**, but may-or-may-not be able to read indexes generated by version **X-2.N**.

Note: Older releases are never guaranteed to be able to read indexes generated by newer releases. When this is attempted, a predictable error should be generated.

Changes to Runtime Behavior

From time to time, the Lucene Developers may decide that a particular bug fix or enhancement (which does not require or involve and API change) should result in a change to the default runtime behavior in some Lucene functionality because the "new" behavior is deemed "better" or more "correct" then the existing behavior. These changes will not necessarily be backwards compatible for some Lucene clients who may have dependencies or expectations on the "old" behavior.

If a runtime behavior change is included in a release, this will be clearly noted in the CHANGES.txt file, along with an recommendations about handling the change in your application.

Contrib and Modules

"All contribs are not created equal."

The compatibility commitments of a contrib package can vary based on it's maturity and intended usage. The README.txt file for each contrib should identify it's approach to compatibility. If the README.txt file for a contrib package does not address it's backwards compatibility commitments users should assume it does **not** make any compatibility commitments.

You should also consider Modules individually - a benchmark module might not make the same promises that a grouping module does.