Primordial Class List

The primordial class list indicates which classes should be compiled and baked into the boot image. The bare minimum set of classes needed in the primordial list includes;

1. All classes that are needed to load a class from the file system. The class may need to be loaded as a single class file or out of a jar. Failing this there will be an infinite regress on the first class load.
2. All classes that are needed by the baseline compiler to compile any method. Failing this we regress when attempting to compile a method so we can execute it.
3. Enough of the core VM services and data structures, and class library (java.*) to support the above. This includes threading, memory management, runtime support for compiled code, etc.

For increased performance and decreased startup time it is possible to include extra classes that are expected to be needed. i.e. the optimizing compiler or the adaptive system. There are some pieces of these components that would be awkward to load dynamically (there's a core subset of the opt compiler, the classes in the org.jikesrvm.compilers.opt.runtimesupport packages, the must be loaded and fully compiled before any opt-compiled code can be allowed to executed), but it's theoretically possible to do so.

If you took a full closure of the classes referenced by things that have to be in the bootimage you'd actually end up with a lot more in the bootimage than we currently have. The culprit here would I think mainly be java.* classes that we need in the bootimage, but only use in restricted ways, so we don't actually have to drag in everything they depend on to meet the "real" constraints of what has to go in the bootimage. It is unknown how much difference there is between hand-crafted include lists and what an automated tool would discover.