OpenJDK
Installing
Contributing
Sponsoring
Developers' Guide
Vulnerabilities
JDK GA/EA Builds
Mailing lists
Wiki · IRC
Bylaws · Census
Legal
Workshop
JEP Process
Source code
Mercurial
GitHub
Tools
Git
jtreg harness
Groups
(overview)
Adoption
Build
Client Libraries
Compatibility & Specification Review
Compiler
Conformance
Core Libraries
Governing Board
HotSpot
IDE Tooling & Support
Internationalization
JMX
Members
Networking
Porters
Quality
Security
Serviceability
Vulnerability
Web
Projects
(overview, archive)
Amber
Audio Engine
CRaC
Caciocavallo
Closures
Code Tools
Coin
Common VM Interface
Compiler Grammar
Detroit
Developers' Guide
Device I/O
Duke
Font Scaler
Galahad
Graal
Graphics Rasterizer
IcedTea
JDK 7
JDK 8
JDK 8 Updates
JDK 9
JDK (..., 21, 22)
JDK Updates
JavaDoc.Next
Jigsaw
Kona
Kulla
Lambda
Lanai
Leyden
Lilliput
Locale Enhancement
Loom
Memory Model Update

JEP 11: Incubator Modules

Authors	Chris Hegarty, Alex Buckley
Owner	Chris Hegarty
Type	Informational
Scope	JDK
Status	Active
Discussion	jdk dash dev at openjdk dot java dot net
Effort	S
Duration	S
Reviewed by	Alan Bateman, Alex Buckley, Brian Goetz, Paul Sandoz
Endorsed by	Brian Goetz
Created	2016/11/16 09:17
Updated	2019/12/05 17:14
Issue	8169768

Summary

Incubator modules are a means of putting non-final APIs and non-final tools in the hands of developers, while the APIs/tools progress towards either finalization or removal in a future release.

Goals

- Enable JDK Release Projects to distribute a limited set of APIs and tools that not final and complete, and which would benefit from developer or user feedback. This will reduce the chance of costly mistakes in the Java SE Platform and the JDK.

Non-Goals

- It is not a goal to define a general-purpose mechanism to distribute arbitrary non-JDK modules.
- It is not a goal to define the complete lifecycle of a module, API, or tool.
- It is not a goal that every feature being developed for the JDK be, at some point in its lifecycle, incubated (though that may be desirable).
- It is not a goal to define a mechanism for incubating language or VM features, but any such future mechanism should use incubator modules for their relevant APIs.

Motivation

Many Java SE Platform APIs, supported JDK APIs, or widely-useful JDK tools would benefit from spending a period of time in a JDK Release Project prior to being standardized in the JCP or otherwise being deemed stable. Being in a JDK Release Project, and thus in downstream binaries such as those at jdk.java.net, makes it easier for interested parties outside of the immediate OpenJDK Community to use the new feature. Experience gained and fed back through the usual channels such as blogs, mailing lists, outreach programs, and conferences can then be acted upon before finalizing, or else removing, the feature in a future Release Project.

Description

An *incubating feature* is an API or a tool, of non-trivial size, that is under development for eventual inclusion in the Java SE Platform or the JDK. The API or tool is not yet sufficiently proven, so it is desirable to defer standardization or finalization for a small number of feature releases in order to gain additional experience and feedback.

Metropolis
Mission Control
Modules
Multi-Language VM
Nashorn
New I/O
OpenJFX
Panama
Penrose
Port: AArch32
Port: AArch64
Port: BSD
Port: Haiku
Port: Mac OS X
Port: MIPS
Port: Mobile
Port: PowerPC/AIX
Port: RISC-V
Port: s390x
Portola
SCTP
Shenandoah
Skara
Sumatra
Tiered Attribution
Tsan
Type Annotations
Valhalla
Verona
VisualVM
Wakefield
Zero
ZGC

An *incubator module* is a module in a JDK Release Project that offers an incubating feature: the module either exports an incubating API, or contains an incubating tool, or both.

An incubator module is identified by the `jdk.incubator.` prefix in its module name, whether the module exports an incubating API or contains an incubating tool.

An incubating API is identified by the `jdk.incubator.` prefix in its exported package names. An incubating API is exported only by an incubator module.

An incubating tool may use the `jdk.incubator` prefix in the tool name invoked on the command line, but is not required to.

If an incubating API is standardized in the Java SE Platform, or otherwise promoted to some stable mode of existence in the JDK, then its packages will be renamed and then exported from a non-incubator module. Early adopters will at the very minimum need to change their `import` statements, and may need to refactor their uses if the API changes more substantially. Similarly, if an incubating tool is promoted to some stable mode of existence, then early adopters may need to change any invoking scripts in order to account for changes in the tool's functionality and command line interface.

If an incubating API is **not** standardized or otherwise promoted after a small number of JDK feature releases, then it will no longer be able to incubate, and its packages and incubator module will be removed. Similarly, if an incubating tool is not promoted, then its incubator module will be removed.

Process and Evolution

An incubating feature (API or tool) will use the JEP process as is, with the following guidelines:

1. The JEP that defines the feature should make it clear that incubation is planned, and that the result of completing the JEP will be an incubator module which exports the API/contains the tool.
2. An incubating feature, along with any evolutionary API changes, that is later promoted to the Java SE Platform, or some other stable mode of existence in the JDK, should have a new JEP to propose and document that change. For the new JEP, it is permitted, but not required, to deliver the promoted API or tool in its own module; the API or tool may be integrated into an existing standard module or a non-incubator JDK module.
3. If an incubating feature is later removed then no new JEP is necessary. The incubator module that contains the incubating feature may be silently removed. It is recommended that the original JEP be updated to record the fate of the feature.

One incubator module should contain one incubating feature. (If an incubating tool is associated with an API, then that API may be regarded as part of the tool and exported by the incubator module which contains the incubating tool) Dedicating an incubator module to an incubating feature will help prevent the emergence of a "miscellaneous" incubator module that acts as a dumping ground, as has been seen in the past in some parts of the legacy `sun.*` namespace. There is clear ownership of incubating features and their impending, or not, future.

An incubating feature need not be retained forever in the JDK Release Project in which it was introduced, nor in every release of downstream binaries derived from that Release Project. For example, an incubating feature may evolve, or even be removed, between different update releases of a JDK Release Project. Beyond this explicit statement of when evolution is permitted, this proposal deliberately provides no further guidance. Such decisions are best left to the individual feature owner.

An incubator module, and its API/tool, will not remain in the JDK long term, so in some sense they are terminally deprecated at birth. However, it is cleaner to think of incubation and deprecation as completely separate concepts, so the `@Deprecated` annotation and the `@deprecated` JavaDoc tag should *not* be used in incubator modules or their APIs.

Relationship to other modules

Incubator modules must export incubating APIs only, i.e., packages in the `jdk.incubator` namespace. Consequently:

- Incubator modules must not export APIs in the `java.` or `javax.` namespaces governed by the JCP. This distinguishes incubator modules from *standard modules* such as `java.base`.
- Incubator modules must not export supported JDK APIs. This distinguishes incubator modules from JDK modules such as `jdk.compiler` that do export such APIs.
- Incubator modules must not export *critical internal APIs*. Incubator modules have no relationship to the `jdk.unsupported` module.

Standard modules and non-incubator JDK modules must not specify `requires transitive` dependences upon incubator modules, or otherwise expose types exported from incubator modules in their own exported APIs. In exceptional cases, it may be acceptable for standard modules and non-incubator JDK modules to specify `requires` dependences (as opposed to `requires transitive`) upon incubator modules.

Incubator modules can specify `requires` or `requires transitive` dependences upon other incubator modules.

Incubator modules are part of the JDK run-time image produced by the standard JDK build. However, by default, incubator modules are **not** resolved for applications on the class path. Furthermore, by default, incubator modules do **not** participate in service binding for applications on the class path or the module path.

Applications on the class path must use the `--add-modules` command-line option to request that an incubator module be resolved. Applications developed as modules can specify `requires` or `requires transitive` dependences upon an incubator module directly. (Some incubator modules, such as those offering command-line tools, may forego exporting any packages, and instead provide service implementations so that tools can be accessed programatically. It is usually not recommended to require a module that exports no packages, but it is necessary in this case in order to resolve the incubator module, and have its service providers participate in service binding.)

During the JDK build, incubator modules must be packaged into `jmod` files with the `--do-not-resolve-by-default` option of the `jmod` tool, so that they are not in the default set of root modules for the unnamed module. This makes them, in effect, "opt-in". Also, the `--warn-if-resolved=incubating` option must be passed to the `jmod` tool, so that a warning is issued at compile time, link time, and run time if the incubator module is resolved. This warning can be suppressed at compile time, but not at other times.

Incubator modules are granted no default security permissions.

Incubator modules can be operating-system specific.

Integration points

Many new APIs wish to integrate with the `java.*` and `javax.*` APIs of the Java SE Platform, but during incubation, an API must reside `jdk.incubator.` namespace and so "stands apart" from the SE APIs. For example, consider the Streams API: it may have been necessary for a notional `jdk.incubator.stream.Streams` class to provide static factory methods such as `fromList`, `fromSet`, etc., to enable integration with the existing Collections API. While this is a compromise, integration points with the existing SE APIs typically represent a relatively small surface of a new API.

In some cases an incubating feature may be tightly integrated with the Java run-time system and the JVM. In such cases low-level operations can be exposed through *qualified exports* from the appropriate module(s) in the JDK to the incubator module containing the feature. An incubator module that has at least one qualified export from a module in the JDK must be tightly coupled, i.e., its hash must be recorded in the exporting module, and the incubator module is therefore not upgradeable. An incubator module that does not have any qualified exports from a module in the JDK need not be tightly coupled, and may therefore be upgradeable. If underlying incubation support is required in the JVM then it may be necessary to provide a command-line argument, e.g., `-XX:+UnlockExperimentalVMOptions`, or else arrange to enable such support automatically in the presence of the incubator module.

Documentation

An incubating API will have its JavaDoc built as part of the JDK docs build, in the same way as other JDK APIs. An additional directory structure, `jdk/incubator/` will be added so as to group the documentation of incubating features on a per-module basis. All JavaDoc built for incubating APIs will have explicit, obvious, and consistent warnings about the incubating state of the feature, and will caution that the feature will eventually be removed.

Alternatives

Many new features are tightly integrated with the Java run-time system and the JVM. Distributing them independently of the JDK would thus be difficult, if not impossible, since they need to be tightly tied to a particular build of a particular release. Distributing them with the JDK ensures that all the necessary pieces are tightly coupled.

Early-access (EA) binaries of a JDK Release Project or another Project will continue to be offered so that developers can try new features and offer feedback before the Generally Available (GA) release. However, the number of EA downloads has historically been much smaller than GA downloads, because most developers desire the higher level of stability and quality provided by GA releases. As such, EA releases garner only limited attention and generate only limited feedback, yet incubating features are aimed at a wide range of developers and thus seek broad attention and generous feedback. This makes the EA release of a feature an inadequate alternative to the GA release of an incubating feature.

Testing

The code comprising an incubating feature should be tested as any other within the JDK. Specific feature testing requirements should be outlined by the JEP delivering the incubating feature.

Risks and Assumptions

The obvious risk with an incubating feature is that someone's code or script will come to depend upon it and then be "broken" when run on a later release, in which the incubator feature's module has been removed. This risk is mitigated by ensuring that incubating features are *opt-in*, i.e., by not resolving incubator modules by default, and by issuing warnings, in all phases, when incubator modules are resolved.