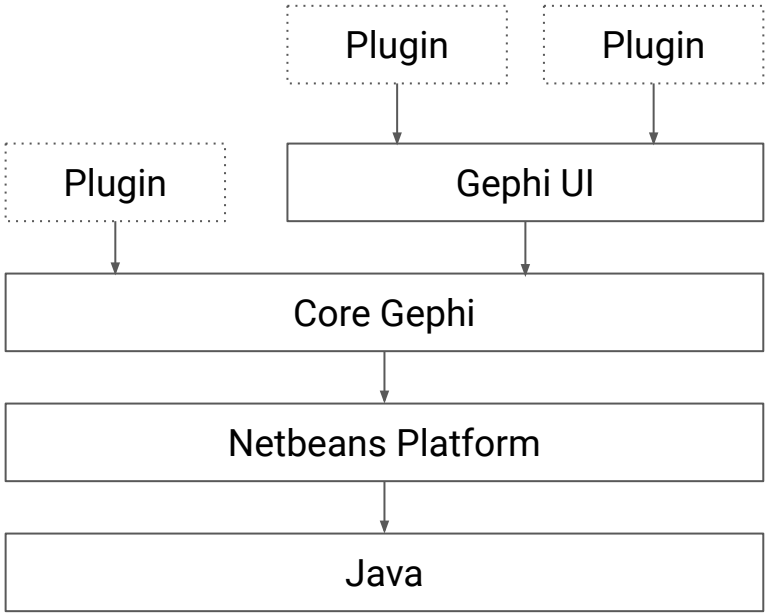
Introduction to Gephi

Codebase and Development

AGENDA

- Introduction
  - Architecture at a high level
  - Principles
  - Technologies
- APIs
  - Main APIs
  - API vs SPI
  - Lookup in practice
  - Future
- Code structure
  - Module and Package conventions
  - Controller and Models
  - Repository structure
- Building and Releasing
  - Building Architecture
  - Releasing Gephi
  - Local Development

Architecture



Principles

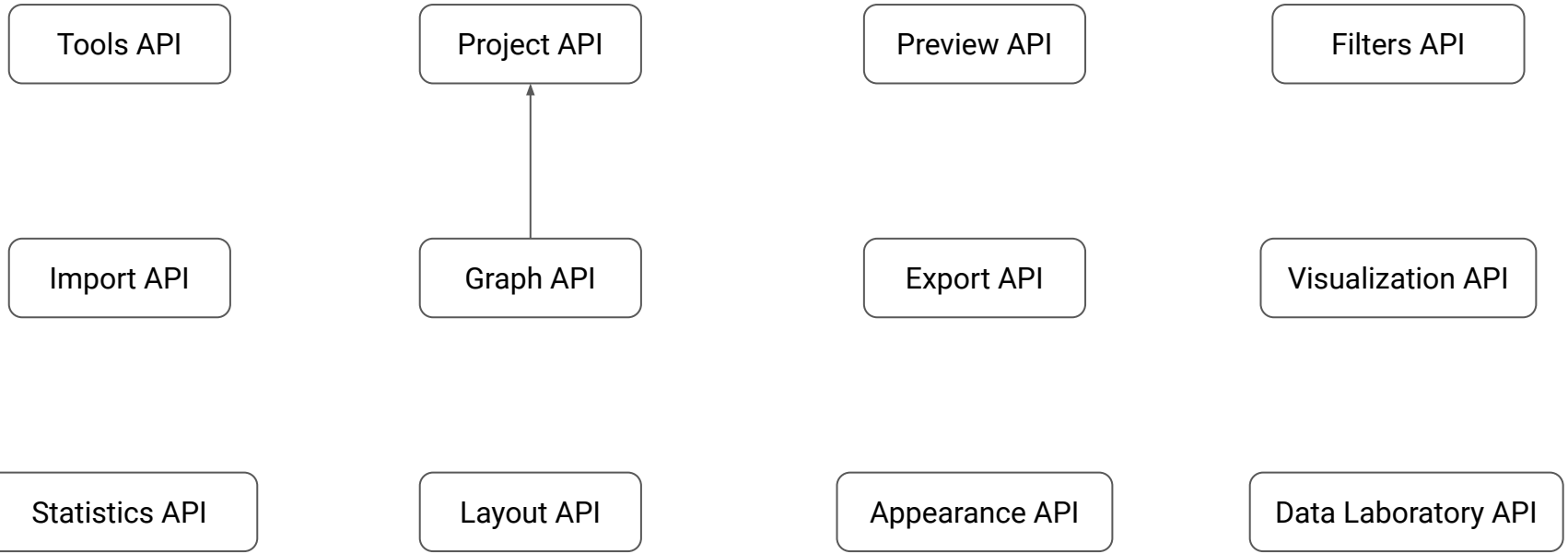
- **Modularity** - Design to be extensible
  - Code broken down in 63 modules
  - Define dependencies between modules
  - Plugins are nothing else than modules
- **Multithreaded** - Non-blocking user interface
  - Long tasks can run in the background
  - Graph structure protected via locking
  - UI is reactive to data changes

Technologies

- Java (JDK11)
- Netbeans Platform
  - Module system
  - Docking framework
  - Auto-update
- UI
  - Swing
  - OpenGL
  - Java2D
- Build and CI/CD
  - Maven
  - JUnit
  - GitHub Actions

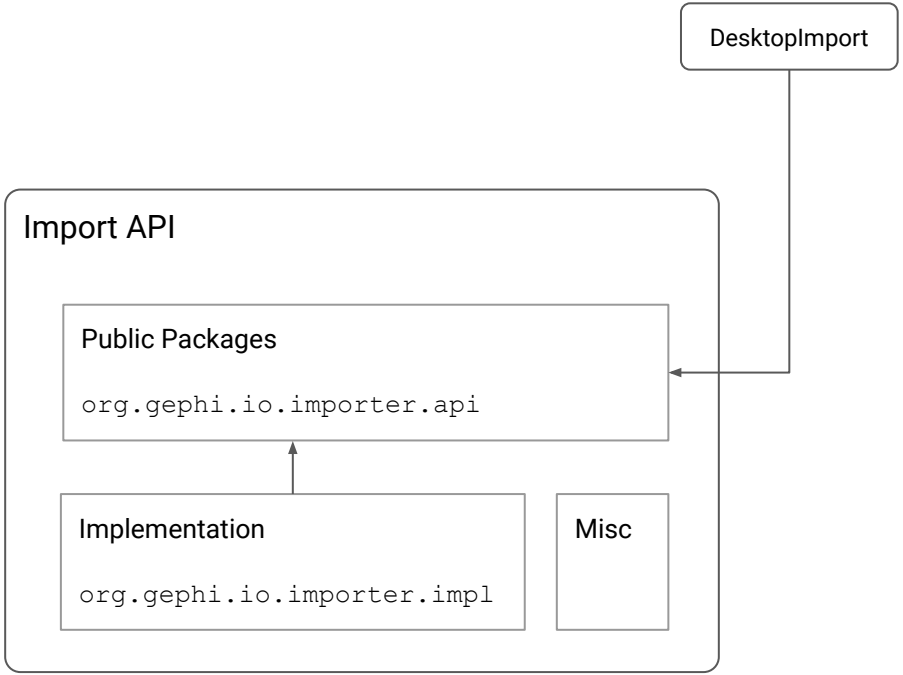
APIs

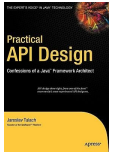
Main APIs


Tools API
Project API
Preview API
Filters API
Import API
Graph API
Export API
Visualization API
Statistics API
Layout API
Appearance API
Data Laboratory API

APIs

- Defined as plain Java interfaces
- Public packages
- *Stable* vs *Under Development*
- Backward compatibility
- Documented via Javadoc



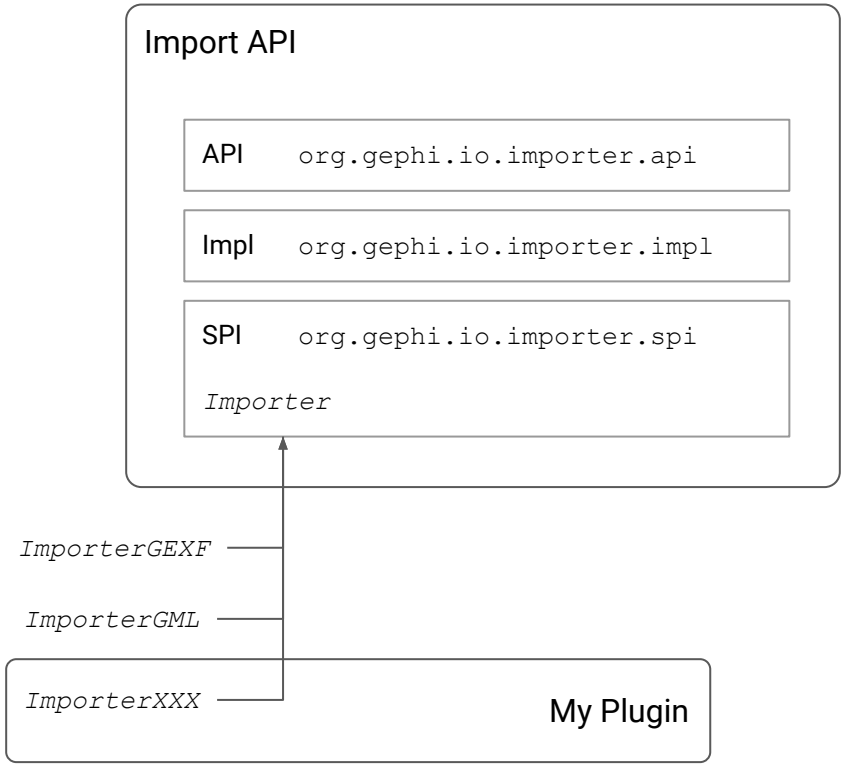


[Practical API Design: Confessions of a Java Framework Architect]

Gephi

SPI

- *Service Provider Interface*
- SPIs meant to be extended
- Same properties than APIs
- But never mixed with APIs



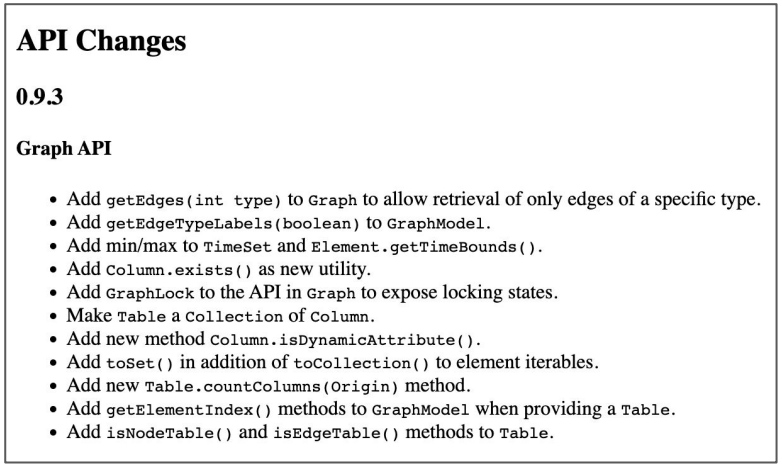
SPIs

Import SPI	File, Database and Wizard importers
Layout SPI	Layout algorithms
Statistics SPI	Other algorithms
Tools SPI	Tools (Menubar)
Project SPI	Persistence Providers
Export SPI	File exporters (graph and graphics)
Filters SPI	Filters
Preview SPI	Preview Builders and Renderers
Generator SPI	Generators (similar to Importer)
Data Laboratory SPI	Manipulators
Appearance SPI	Transformers (Ranking, Partition)
Visualization SPI	Renderers (future viz engine)

APIs vs SPIs

- *Let's recap!*
- APIs offer functionalities to other modules
- SPIs are meant to be extended
- Plugins always extend an SPI
- Both should be clearly documented

API/SPI Changes are documented in the Javadoc overview page



API Changes

0.9.3

Graph API

- Add `getEdges(int type)` to `Graph` to allow retrieval of only edges of a specific type.
- Add `getEdgeTypeLabels(boolean)` to `GraphModel`.
- Add min/max to `TimeSet` and `Element.getTimeBounds()`.
- Add `Column.exists()` as new utility.
- Add `GraphLock` to the API in `Graph` to expose locking states.
- Make `Table` a `Collection` of `Column`.
- Add new method `Column.isDynamicAttribute()`.
- Add `toSet()` in addition of `toCollection()` to element iterables.
- Add new `Table.countColumns(Origin)` method.
- Add `getElementIndex()` methods to `GraphModel` when providing a `Table`.
- Add `isNodeTable()` and `isEdgeTable()` methods to `Table`.

Resources: https://netbeans.apache.org/wiki/DevFaqApiSpi.asciidoc

Lookup in Practice

- Most important Netbeans Platform utility
- Find the singleton service

```
ProjectController pc = Lookup.getDefault().lookup(ProjectController.class);
```

- Find all implementations of an SPI

```
for(Renderer renderer : Lookup.getDefault().lookupAll(Renderer.class)) { }
```

- Relies on SPI annotation

```
@ServiceProvider(service = Renderer.class)
public class NodeRenderer implements Renderer { }
```

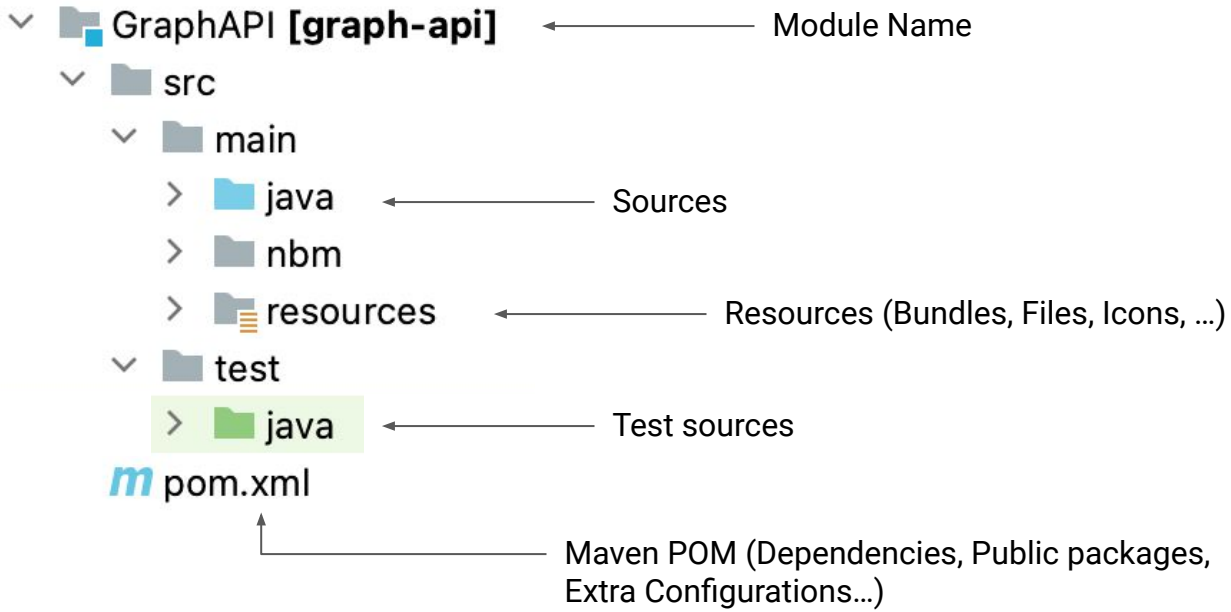

Resources: https://netbeans.apache.org/wiki/DevFaqLookup.asciidoc

APIs Future

- What we have achieved and want to preserve
  - For each feature, we have a clean, stable and documented API
  - Implementations can be replaced
  - New features can be added (via plugins)
  - Can be used in command-line tools (no UI needed)
  - Modules can be reused in other projects (everything is on Maven Central)
- Semantic Versioning
  - Patch (0.9.x) - minimal API changes
  - Minor (0.x.x) - API additions and occasionally breaking compatibility when necessary
  - Major (x.x.x) - Rewriting APIs from scratch with major changes

Code Structure

Anatomy of a module



Module name conventions

63 modules in total

- **ImportAPI** → Defines the APIs & SPIs and implement APIs. — **17 modules**
- **ImportPlugin** → SPI implementations — **11 modules**
- **ImportPluginUI** → SPI implementations (UI only) — **6 modules**
- **DesktopImport** → UI code — **15 modules**

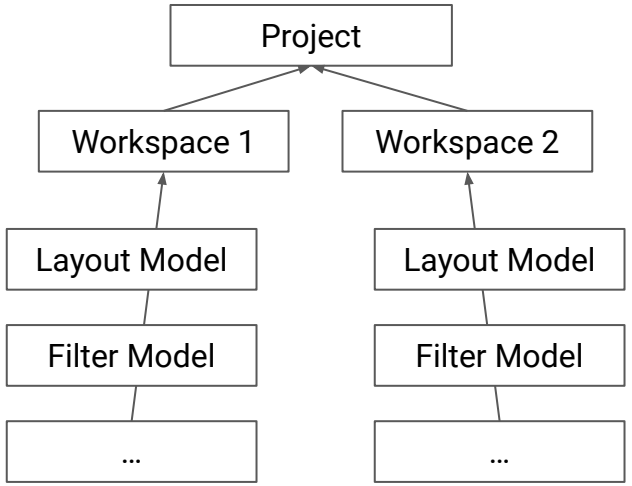
49 modules

- **Package name conventions**

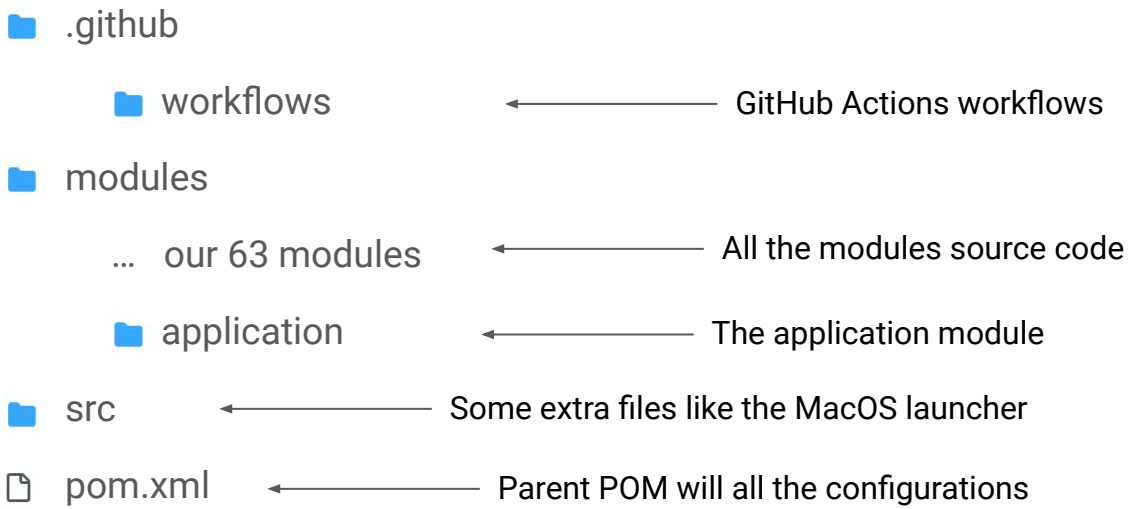
API	org.gephi.NAME.api
API Implementation	org.gephi.NAME
SPI	org.gephi.NAME.spi
SPI Implementation	org.gephi.NAME.plugin
SPI Implementation UI	org.gephi.ui.NAME.plugin
UI	org.gephi.desktop.NAME

Controller and Models

- Model ~~View~~ Controller Pattern
- Controllers
  - Singleton, found via Lookup
  - Entry point for any Model alteration (setters)
  - Retrieve the model
- Models
  - Contains all the states and data
  - One model per Workspace
  - Source for Persistence Providers



Repository


.github
workflows
GitHub Actions workflows
modules
... our 63 modules
All the modules source code
application
The application module
src
Some extra files like the MacOS launcher
pom.xml
Parent POM will all the configurations

Building and Releasing

Building Architecture

- master branch → commit →  build and test
  - feature branch → PR
- release branch (e.g. 0.9.7) → commit →  release
  - Build and test
  - Produce and deploy artifacts to Maven central (example)
  - Update AutoUpdate site

Same process for development and final releases.

If version ends with "-SNAPSHOT" it's a development release.

Releasing Gephi

- Entirely Automated
  - Trigger: Commit to a release branch with POM files without "-SNAPSHOT"
  - Produces the final installers and deploys them to Maven Central (link)
  - Updates AutoUpdate site
- Only manual steps
  - Manipulate the version in the *pom.xml* files
  - Create the release on GitHub

Behind the scene

- Matrix build with one job per OS and Architecture (four currently)
- Embedded JRE
- Workflow depends on OS
  - Mac OS: Code signing and App notarization
  - Windows: Create Installer with InnoSetup

Example

Local Development

- Netbeans IDE and IntelliJ IDEA are officially supported IDEs
- Project recognised as a multi-module Maven project
- Running and Debugging locally (including Unit tests)
- Scenarios
  - **New to the code:** Run a complete build (Maven parallel builds OK)
  - **Make changes to a module:** Rebuild this module only
  - **Run locally:** Always rebuild the *application* module as well

Appendix

Resources

- Netbeans Platform
  - Netbeans APIs Documentation
  - Source code
  - FAQ
  - Documentation
  - Tutorials
- Conventions
  - Code Style
- Localization
  - Documentation
- Coding via examples
  - Plugins Bootcamp
  - Toolkit Demos