API Tooling in the Eclipse
Overview

• The need for tooling
• Tooling features
• Tooling in action
• Future work
• Summary
• Q&A
The need for tooling
Define API

• APIs are published contracts
  - Producers specify the contracts
    ▪ Honor contracts release after release
    ▪ Evolve the contracts for enhancements and fix problems
  - Consumers adhere to the contracts
    ▪ Implement the contracts per specification
    ▪ Use provided implementations per contract specifications
API Dimensions

API Evolution (Producer)

API Use (Consumer)

Behavior remains constant over time as APIs evolve and grow.
What have we done as plug-in developers?

• We use Javadoc™ to document contracts
  ◦ This class is not intended to be instantiated or subclassed by clients.
  ◦ Has no effect if not read
  ◦ Inconsistent wording, placement, and use

• Use component.xml to specify APIs
  ◦ Out of date, not maintained
  ◦ Why? Because there’s no tooling and it’s separate from the code

• We use package names to categorize as public/internal, but all packages are exported

• We use OSGi package exports to limit visibility, but does not prevent access to internal types

• We use Eclipse’s “access rules” to discourage use, but this can be turned off
More things we do…

• Manually examine API changes or use some external tool before a release
  ♦ Changes between releases can be significant
  ♦ Validation is time consuming and error prone
  ♦ Lost development time (due to checking and bugs found)
Disclaimer

• API design is still your problem
  - Designing quality APIs
  - Ensuring consistent behavior

• But there are things that tooling can with…
API Tooling Features
API Tooling to the Rescue!

• Reports
  ✷ Illegal API use
  ✷ Binary incompatibility relative to a baseline
  ✷ Incorrect bundle version numbers
  ✷ Missing or malformed @since tags
  ✷ Leakage of non-APIs types inside APIs

• Tightly integrated toolset in the Eclipse SDK
  ✷ Currently limited to Plug-in projects/OSGi bundles
  ✷ Runs as a builder (auto-build, incremental and full builds)
  ✷ Immediate feedback as you develop and use APIs
Specifying API Contracts

• Use Javadoc tags
  • E.g. @noimplement, @noextend, @noreferrer, @noinstantiate

• Benefits
  • Contracts live with the code for producers and consumers
  • Content assist helps developers
  • Available for projects that are not using 1.5 annotations
  • Restrictions appear in published Javadoc APIs in a standard way
  • Tools can process tags
Example of Published API
API Usage

• Once the APIs are specified, the user needs to make sure that he/she is using them appropriately.

• API usage is flagging any kind of illegal usage of API: reference to an API that is not supposed to be referenced, implementation of an interface that is not supposed to be implemented, ....

• A consequence of wrong API usage is a potential binary incompatible change error.
Validating Binary Compatibility

• Evolving APIs such that they are backwards compatible with existing binaries
   http://wiki.eclipse.org/index.php/Evolving_Java-based_APIs
   It is easy to get it wrong
   Now the tooling takes care of this

• The user simply specifies an API baseline
   Generally this means pointing to the previous release (N – 1)
Bundle version number management

- The tooling takes care of letting the user know when the minor or major version of a bundle should be changed according to rules described in the document:
  - A new API that is not a breaking change requires the minor version to be incremented
  - A new API that is a breaking change requires the major version to be incremented
- No support for the micro version for the initial release.
  - Technically speaking, this version should be changed as soon as any modification is made in the source code: comment change, method body change,…
API Tooling in Action (simulation of bug 191231/191232)
API Tooling Parts
API Profile and API Components

• These are the two major pieces used by the API tooling tools to make diagnosis on the code
• An API profile can be seen like a PDE target platform.
• An API profile is a collection of API components.
• It is initialized using an Eclipse SDK installation plugins directory.
• Inside the IDE, a profile corresponding to the workbench contents is automatically created.
API Description

- This contains the specific restrictions for all the API types of an API component
- It is kept up-to-date inside the workbench by using resource listeners
- It is exported inside the binary bundles or generated at build time using an ant task.
Filtering of API Problems

• Each component can define a problem filter
• The filtering can be used to remove from reports known breakages.
• For example, an API breakage has been approved by the PMC and you don’t want to get it reported for each build.
• The problem filter is also part of the binary plug-in
Two aspects: IDE and build process

• Each feature is available from within the IDE or inside the build process.
• The IDE support is required to help the Eclipse developer while the code is written
• The build process support is required to provide feedback during the Eclipse build. This also allows other projects to use it inside their builds.
Build support

• Addition of new ant tasks:
  - Generation of .api_description file
  - Comparison of SDK drops: binary compatibility, api usage reports
• Integration inside the Eclipse builds (headless mode)
• Integration inside ant build (no Eclipse running)
API Error/Warning Preferences
API Setup Wizard

API Tooling Setup Wizard

Configure projects for API usage and binary compatibility checks.

Select the Java projects to update.

Projects:
- org.eclipse.test.performance.data

- [ ] Delete component.xml after update is complete

Changes to be performed:
- Workspace
- org.eclipse.test.performance.data
- Add API nature and API analysis builder

No preview available

Finish  |  Cancel
API Profile Preferences

![Preferences window with API Profiles]

Profiles
Add, remove or edit API profiles. The default (checked) profile is used for binary compatibility analysis.

Profiles:
- Eclipse (current)
- Eclipse 3.2.2
- Eclipse 3.3.1
- Eclipse 3.3.2

Options
Missing default API profile:
- Warning

Restores Defaults
Apply
OK Cancel
API Profile Wizard
Future work
To be done...

• Handling of package versioning
• Support extended to support more than just bundles
  ♦ Pure Java™ projects
  ♦ Plug-in extension points
• Detect illegal use of system libraries with regards to the execution environment set for a project
  ♦ i.e. referencing J2SE™-1.5 code when set to J2SE-1.4
• Improve integration with rel-eng build reporting
• Determine compatible version range of required bundles
• And what you might suggest...
Summary
API Tooling today

• Help you to define your API restrictions
• Keep a consistent and standard presentation of API restrictions
• Detect binary breakage between a baseline and the current version
• Detect wrong API usage
• Detect wrong @since tags and inconsistent bundle versioning
Links

• Wiki
  - [http://wiki.eclipse.org/Api_Tooling](http://wiki.eclipse.org/Api_Tooling)

• Bugzilla
  - [https://bugs.eclipse.org/bugs/enter_bug.cgi?product=PDE](https://bugs.eclipse.org/bugs/enter_bug.cgi?product=PDE)
Q & A