eclipseCON™ 2008

API Tooling in the Eclipse™ SDK

Olivier Thomann
Darin Wright
Michael Rennie
IBM Rational™

March 17th, 2008

Overview

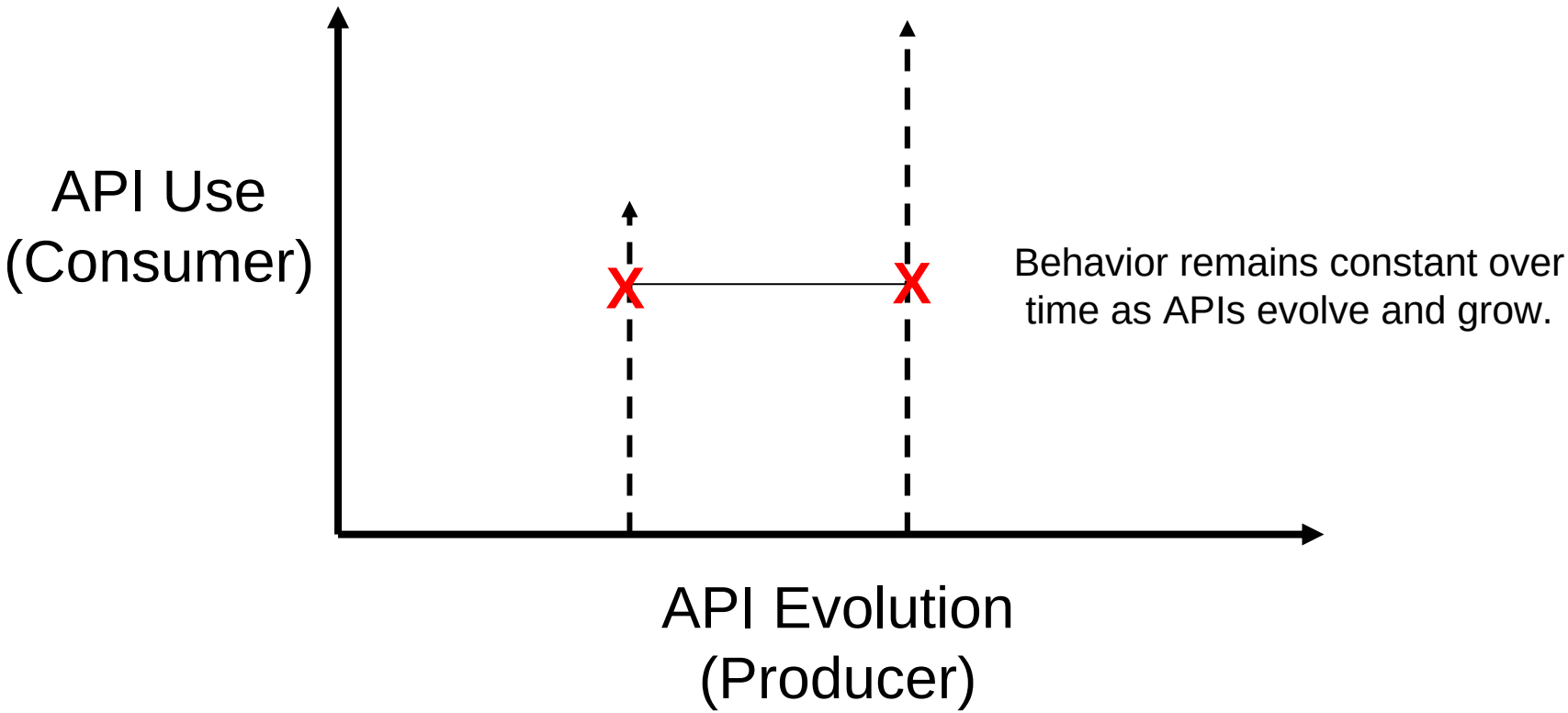
- The need for tooling
- Tooling features
- Tooling architecture
- Future work
- Summary
- Q&A

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



What do we do today 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 API restrictions
  - Out of date, not maintained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 today…

- 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)

Some Existing Tools

- Diff
  - used to track API addition after the API freeze for the past releases.
- PDE incubator API tooling
  - used snapshots to be able to diagnose API breakages
- AUS (API Usage Scanner)
  - IBM tool to detect wrong usage of APIs
    http://www.alphaworks.ibm.com/tech/aus
- JDiff (used by TPTP)
  - http://www.jdiff.org/
- Open-source tool for jar comparison:
  - http://sab39.netreach.com/Software/Japitools/42/ (GPL)
- WTP tool using component.xml.
  - Error-prone as there is no tooling to maintain the component.xml file.

Disclaimer

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

- But there are things that tooling can with…

API Tooling Features

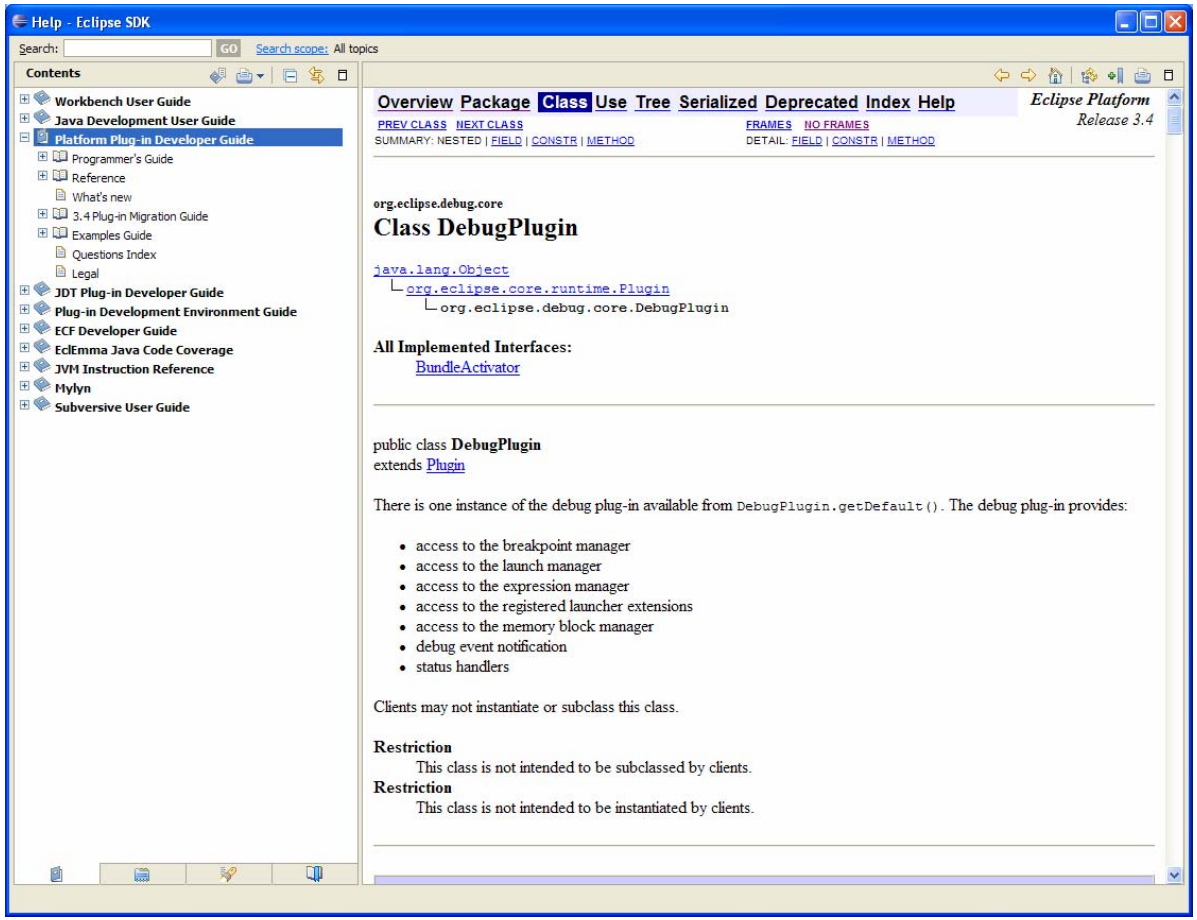
API Tooling to the Rescue!

- 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, @noreference, @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)

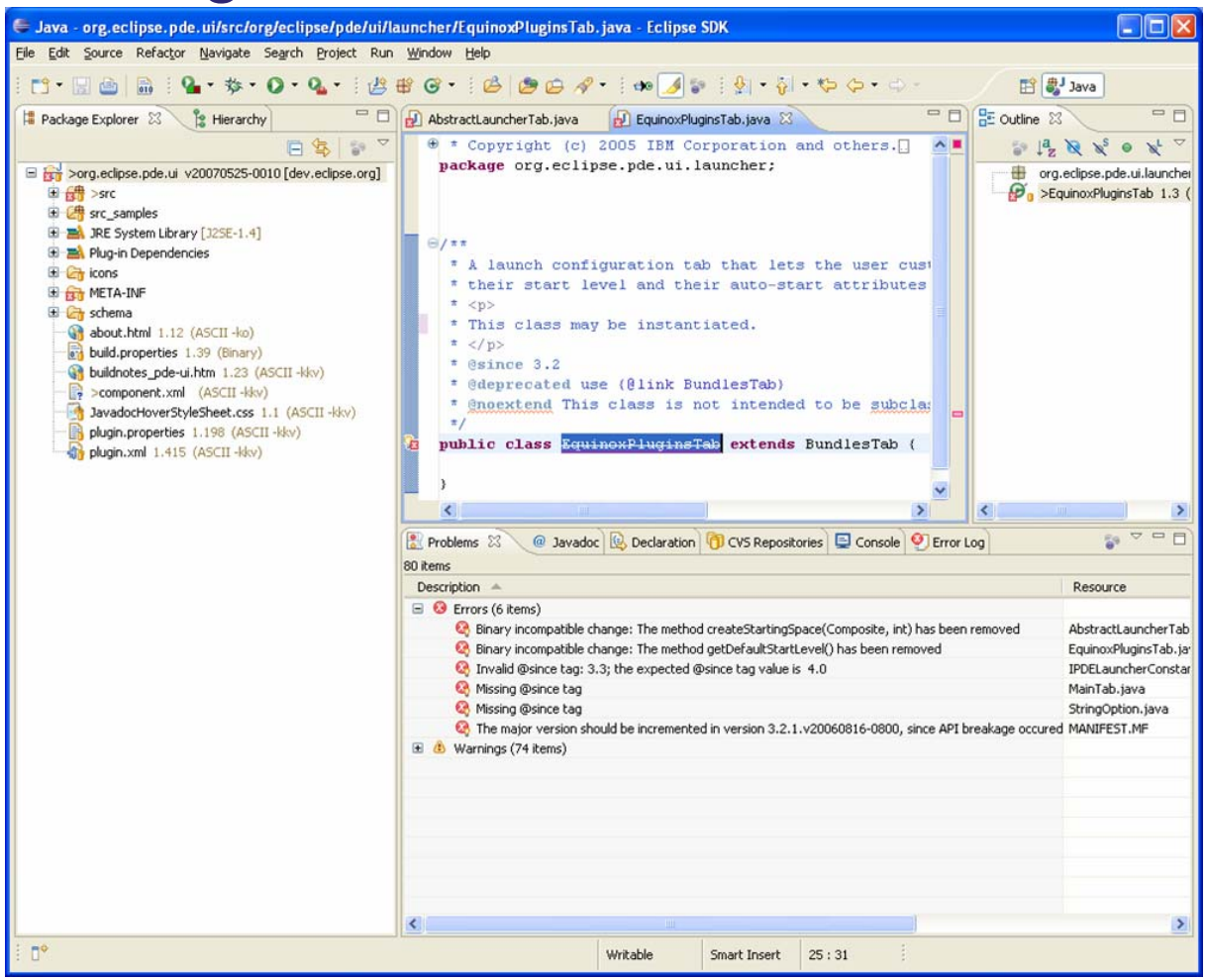
Bundle Version Number Management

- http://wiki.eclipse.org/index.php/Version_Numbering
- 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,…

Detecting API Leakages

- This is a crucial task to define a good API
- It detects the usage of non-API types inside API definition through return types of methods, method parameter types, thrown exceptions and field's type.
- Having such leaks can make your API unusable since the internal types might not be accessible

API Tooling in Action (simulation of bug 191231/191232)



API Tooling Architecture

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.
- An API component maps to an OSGi bundle.
- A profile 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.

Inside Problem Detection

- A search engine finds all references to “restricted” elements
  - Uses ASM to analyse class files
  - Based on binaries (not source)
- A comparator that builds general purpose deltas describing the differences between API profiles
  - finds method additions, removal, …
  - only checks API packages and types

Filtering of API Problems

- Each component can define a problem filter
- The filtering can be used to remove known breakages from reports.
  - 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.

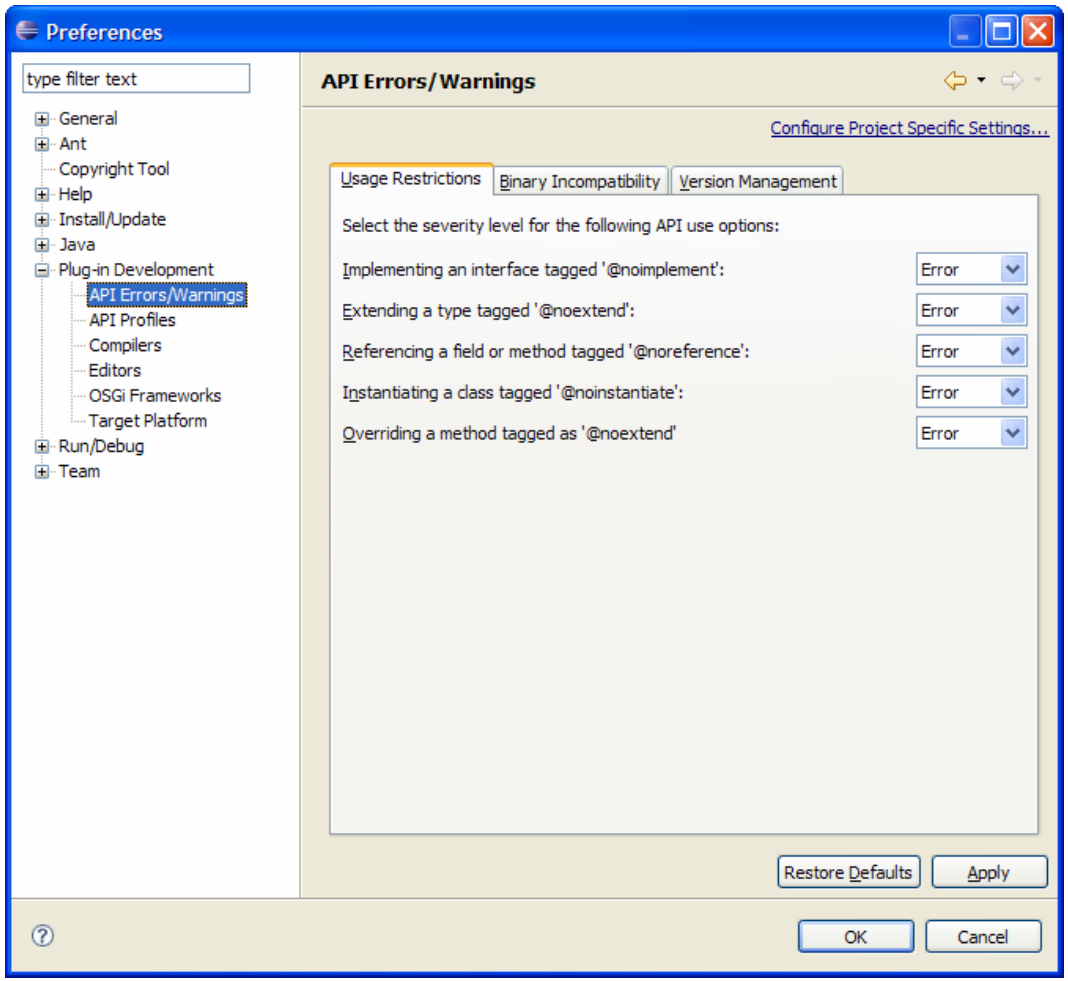
IDE support

- Addition of a new builder
- Addition of a new nature
- Creation of markers for each type of api problems: missing @since tags, binary compatibility issues, api usage, …
- Addition of quickfixes
- Addition of code assist for javadoc tag proposals
- Full build and incremental build support
  - Leverage java builder for incremental dependancy analysis

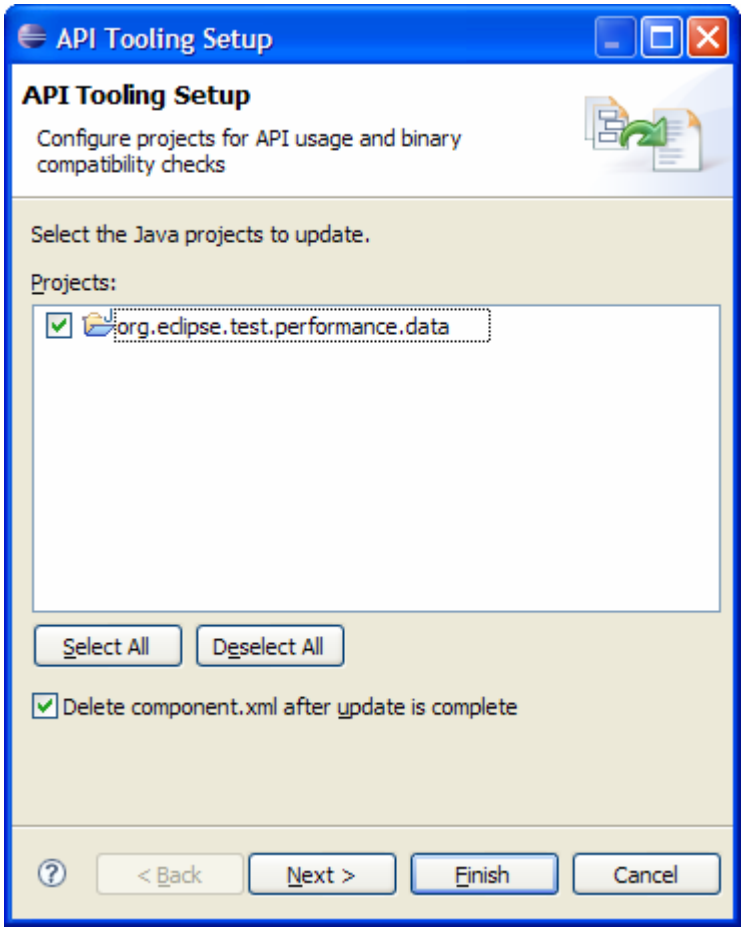
Releng Build support

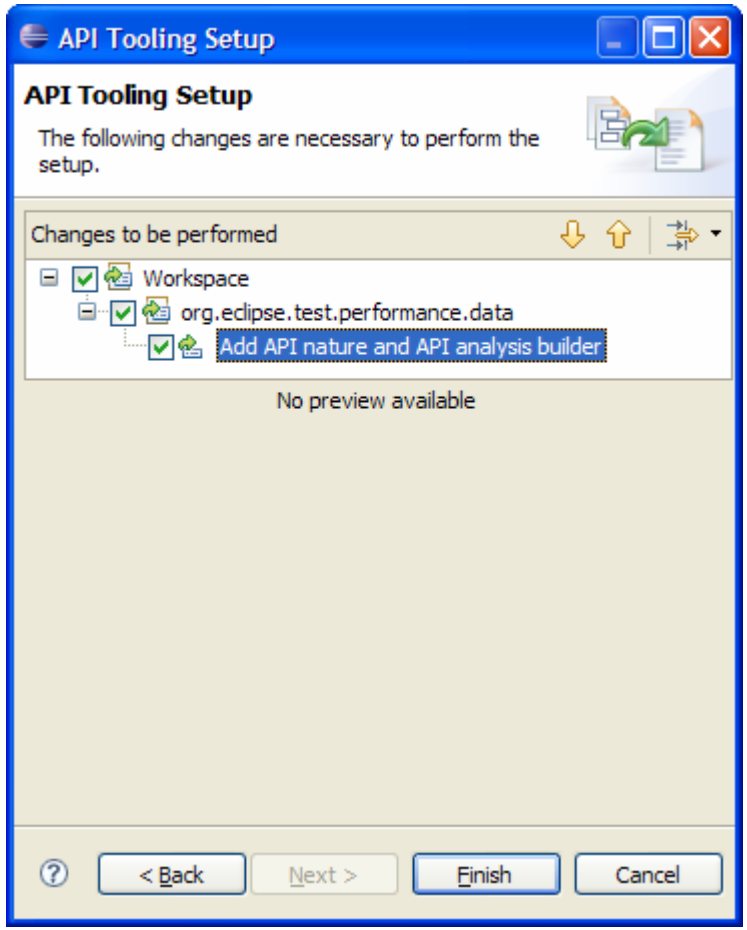
- Addition of new ant tasks:
  - Generation of .api_description file
  - Comparison of SDK drops: binary compatibility, api usage reports
- Integration inside the Eclipse builds (headless mode)
- Integration inside ant build (no Eclipse running)

API Error/Warning Preferences

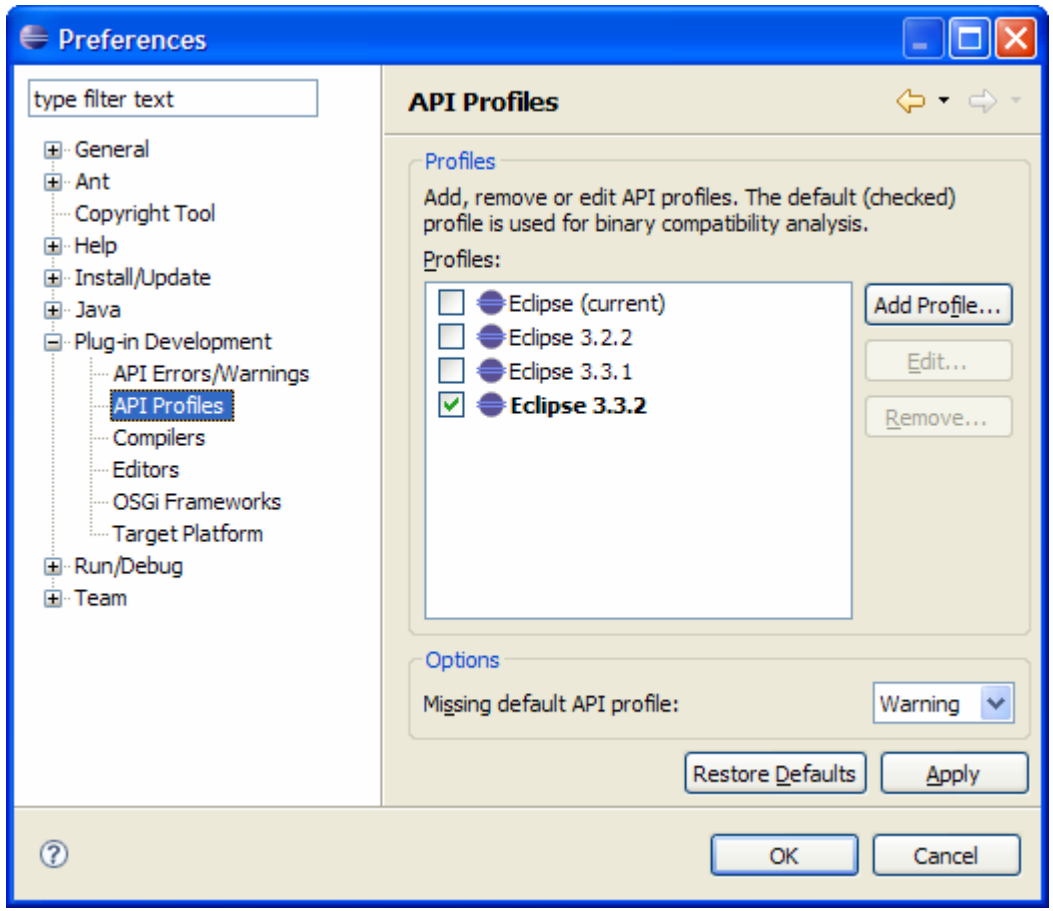


API Setup Wizard

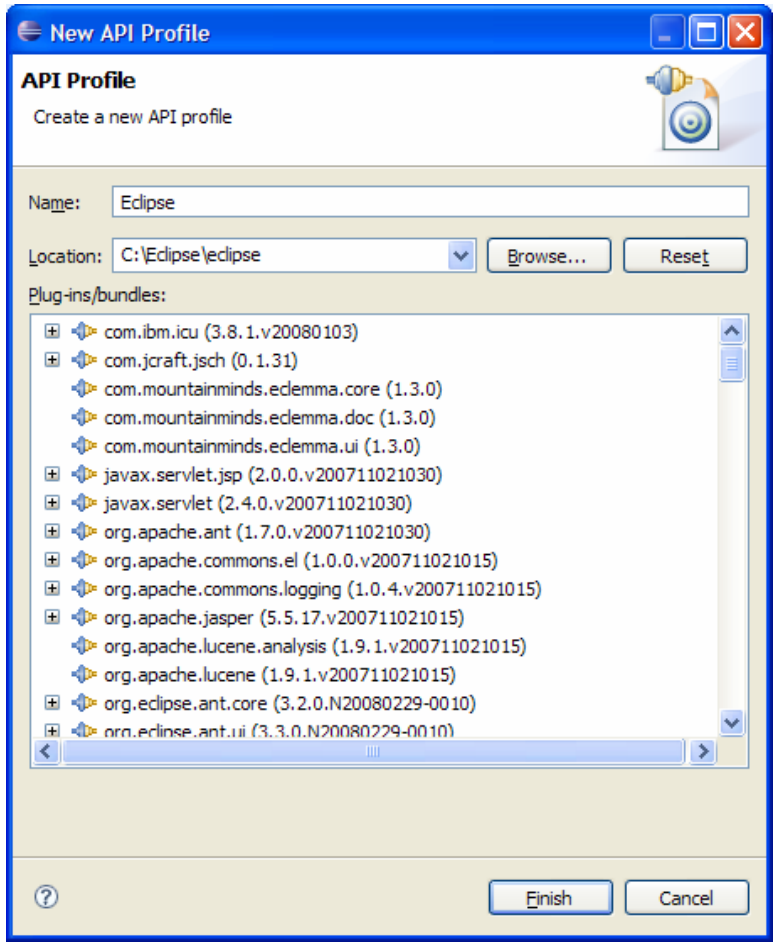




API Profile Preferences



API Profile Wizard



Demos

Stub creator

- API tooling provides an application that can be used to shrink the size of a jar file, a .class file or a set of them (inside a folder)
- A stub is a compressed version of the same .class file in which the tool removes all unnecessary data.
- Several options are provided like preserving the references, visibility filters and compress the stub files.
- A stub version can be used to define an API profile in order to reduce the space used on disk

Future work

To be done post 3.4

- Handling of package versioning
- Support extended to support more than just bundles
  - Pure Java™ projects
  - Consider plug-in extension points
- Global searching inside Eclipse projects
- 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 wrong API usage
- Detect binary breakage between a baseline and the current version
- Detect wrong @since tags
- Detect inconsistent bundle versioning
- Detect API leaking

Links

- Wiki
  - http://wiki.eclipse.org/Api_Tooling
- Bugzilla
  - https://bugs.eclipse.org/bugs/enter_bug.cgi?product=PDE
- Please send me any questions you might have to the API tooling team:
  - Darin_Wright@ca.ibm.com
  - Michael_Rennie@ca.ibm.com
  - Olivier_Thomann@ca.ibm.com

Legal Notices

- Copyright © IBM Corp., 2007-2008. All rights reserved. Source code in this presentation is made available under the EPL, v1.0, remainder of the presentation is licensed under Creative Commons Att. Nc Nd 2.5 license.
- IBM and the IBM logo are trademarks or registered trademarks of IBM Corporation, in the United States, other countries or both.
- Java and all Java-based marks, among others, are trademarks or registered trademarks of Sun Microsystems in the United States, other countries or both.
- Eclipse and the Eclipse logo are trademarks of Eclipse Foundation, Inc.
- Other company, product and service names may be trademarks or service marks of others.
- THE INFORMATION DISCUSSED IN THIS PRESENTATION IS PROVIDED FOR INFORMATIONAL PURPOSES ONLY. WHILE EFFORTS WERE MADE TO VERIFY THE COMPLETENESS AND ACCURACY OF THE INFORMATION, IT IS PROVIDED "AS IS" WITHOUT WARRANTY OF ANY KIND, EXPRESS OR IMPLIED, AND IBM SHALL NOT BE RESPONSIBLE FOR ANY DAMAGES ARISING OUT OF THE USE OF, OR OTHERWISE RELATED TO, SUCH INFORMATION. ANY INFORMATION CONCERNING IBM'S PRODUCT PLANS OR STRATEGY IS SUBJECT TO CHANGE BY IBM WITHOUT NOTICE.

Q & A

