

API Tooling in the Eclipse

Overview

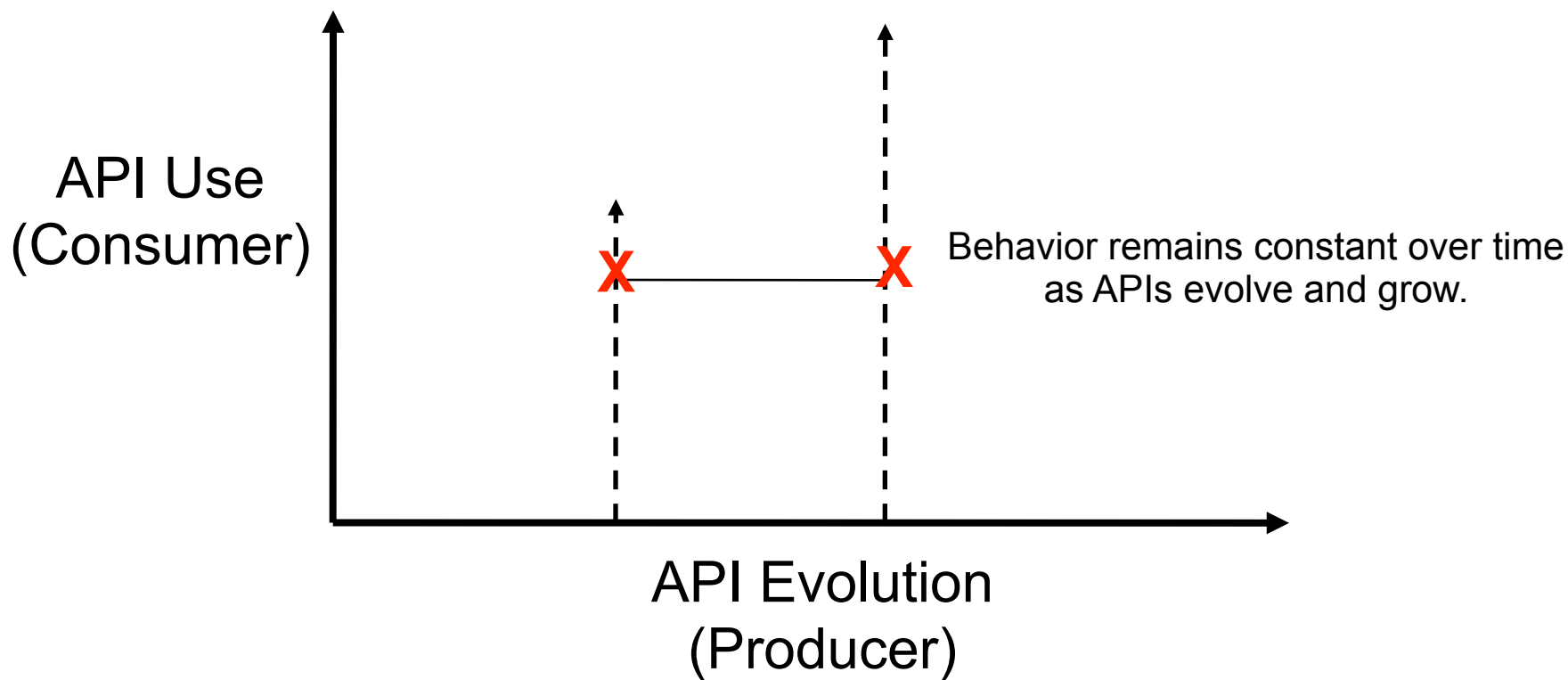
- The need for tooling
- Tooling features
- Tooling in action
- Future work
- Summary
- Q&A

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



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 - ◆ Out of date, not maintained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

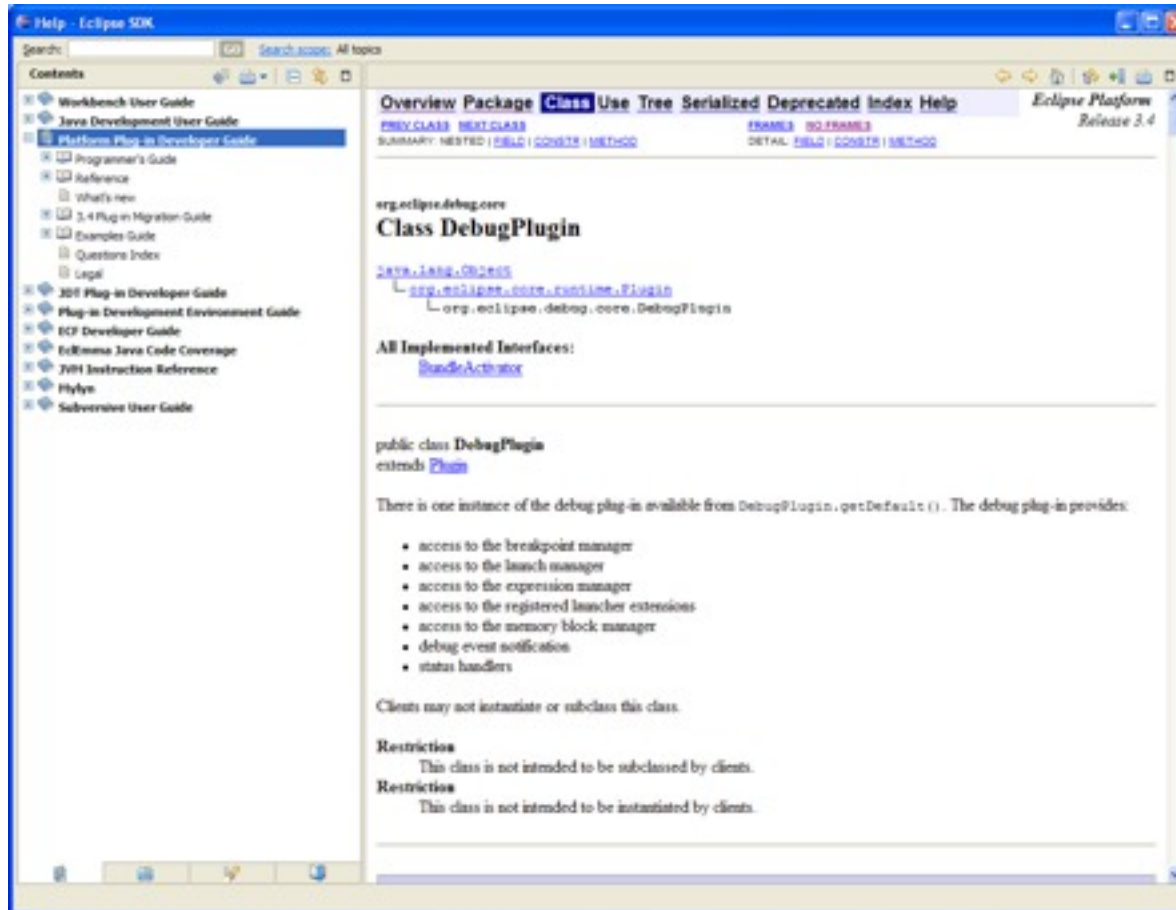
API Tooling to the Rescue!

- 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`, `@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



The screenshot shows the Eclipse IDE Help window for the `org.eclipse.debug.core.DebugPlugin` class. The left sidebar contains a table of contents with 'Platform Plug-in Developer Guide' selected. The main content area displays the class documentation, including navigation links, package information, a class declaration, implemented interfaces, and a list of provided services.

Search: search:acos: All topics

Contents

- Workbench User Guide
- Java Development User Guide
- Platform Plug-in Developer Guide**
 - Programmer's Guide
 - Reference
 - What's new
 - 3.4 Plug-in Migration Guide
 - Examples Guide
 - Questions Index
 - Legal
- JDT Plug-in Developer Guide
- Plug-in Development Environment Guide
- ECF Developer Guide
- Eclemma Java Code Coverage
- JVM Instruction Reference
- Phlyns
- Subversive User Guide

Overview Package **Class** Use Tree Serialized Deprecated Index Help Eclipse Platform
Release 3.4

[PREV CLASS](#) [NEXT CLASS](#) [FRAME](#) [NO FRAME](#)
[SUMMARY](#) [NESTED](#) [CLASS](#) [CONST](#) [METHOD](#) [DETAIL](#) [FIELD](#) [CONST](#) [METHOD](#)

org.eclipse.debug.core
Class DebugPlugin

[JAVA LANG OBJECTS](#)
[org.eclipse.core.runtime.Plugin](#)
[org.eclipse.debug.core.DebugPlugin](#)

All Implemented Interfaces:
[IBundleActivator](#)

public class DebugPlugin
 extends [Plugin](#)

There is one instance of the debug plug-in available from `DebugPlugin.getDefault()`. The debug plug-in provides:

- access to the breakpoint manager
- access to the launch manager
- access to the expression manager
- access to the registered launcher extensions
- access to the memory block manager
- debug event notification
- status handlers

Clients may not instantiate or subclass this class.

Restriction
 This class is not intended to be subclassed by clients.

Restriction
 This class is not intended to be instantiated by clients.

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.

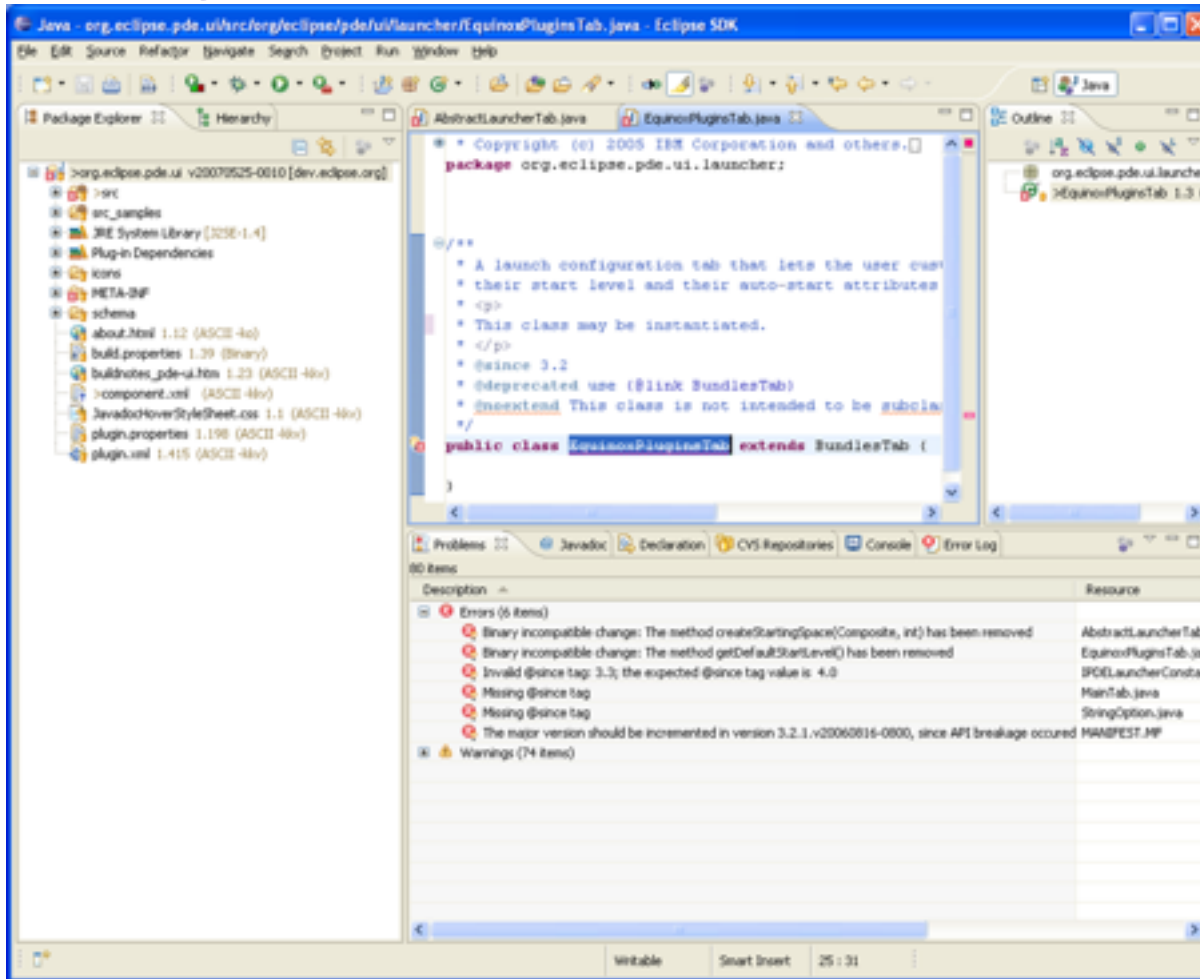
Validating Binary Compatibility

- 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,...

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



The screenshot shows the Eclipse IDE with the following components:

- Package Explorer:** Shows the project structure for org.eclipse.pde.ui, including packages like src, src_samples, JRE System Library, Plug-in Dependencies, icons, META-INF, schema, and various HTML/XML files.
- Editor:** Displays the source code for EquinoxPluginsTab.java. The code includes a package declaration, a class comment, and a public class definition extending BundlesTab.
- Outline:** Shows the class hierarchy for EquinoxPluginsTab.
- Problems:** Lists 80 items, including 6 errors and 74 warnings. The errors are:
 - Binary incompatible change: The method createStartingSpace(Composite, int) has been removed (Resource: AbstractLauncherTab)
 - Binary incompatible change: The method getDefaultStartLevel() has been removed (Resource: EquinoxPluginsTab.java)
 - Invalid @since tag: 3.3; the expected @since tag value is: 4.0 (Resource: SPGLauncherConsta...)
 - Missing @since tag (Resource: MainTab.java)
 - Missing @since tag (Resource: StringOption.java)
 - The major version should be incremented in version 3.2.1.v20060616-0800, since API breakage occurred (Resource: MANIFEST.MF)

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

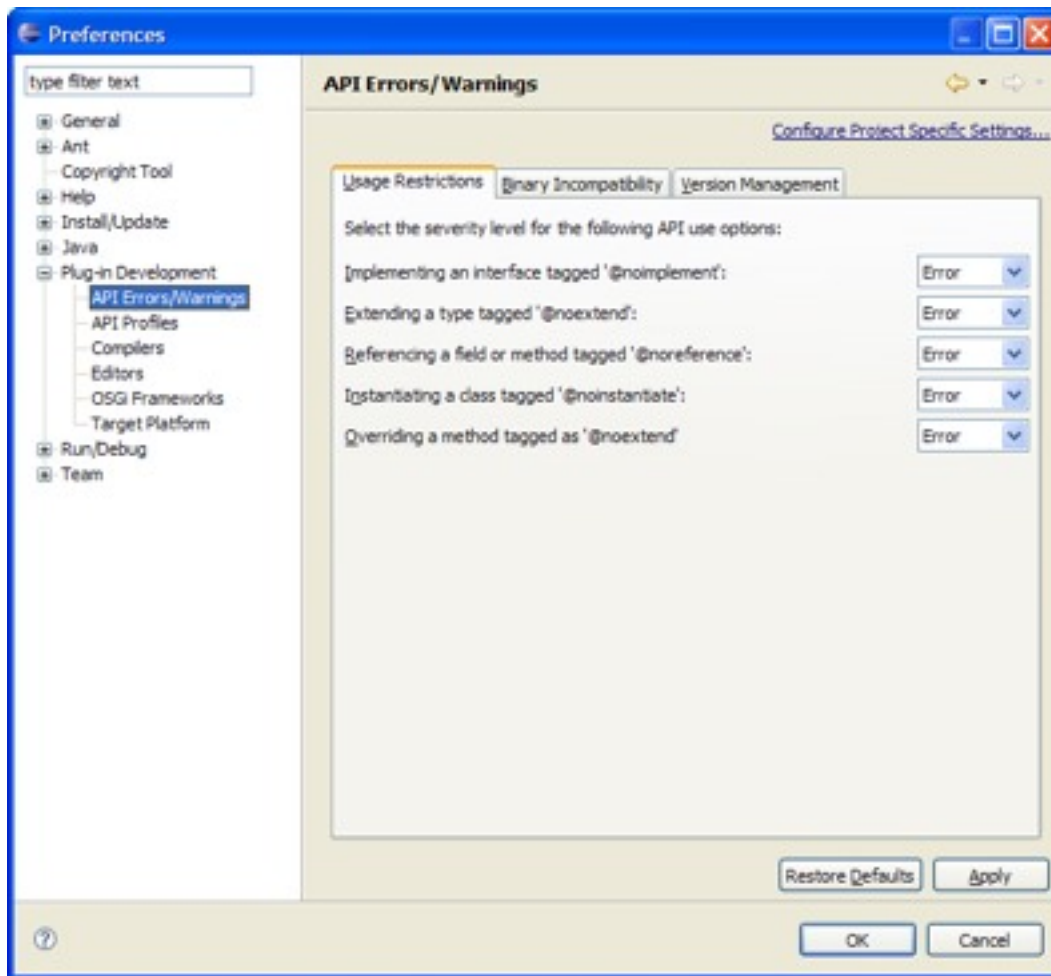
Two aspects: IDE and build process

- 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.

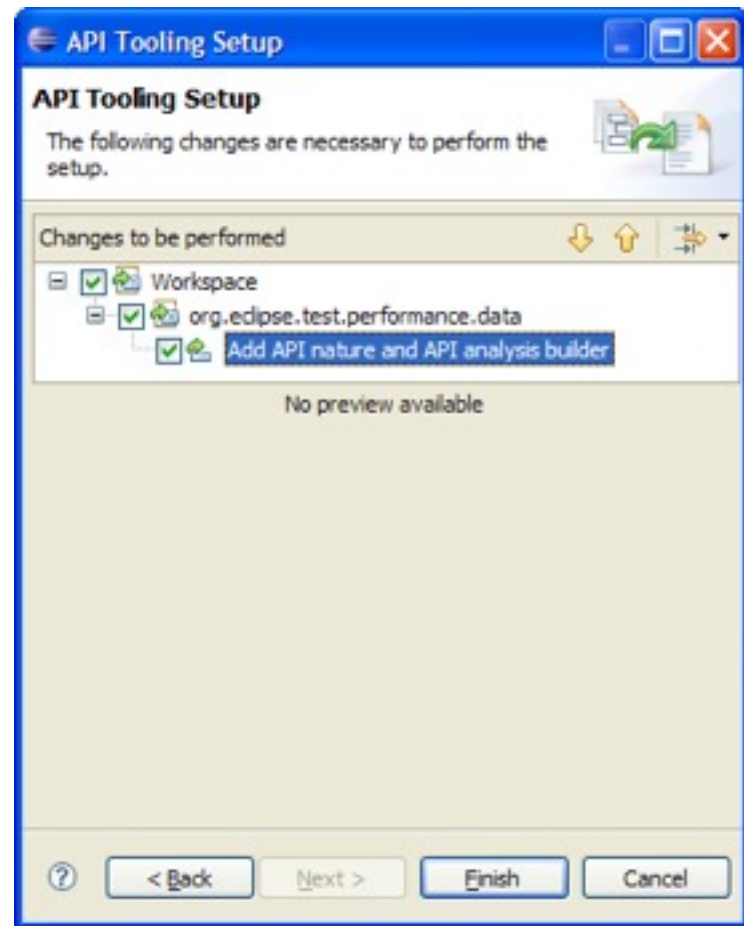
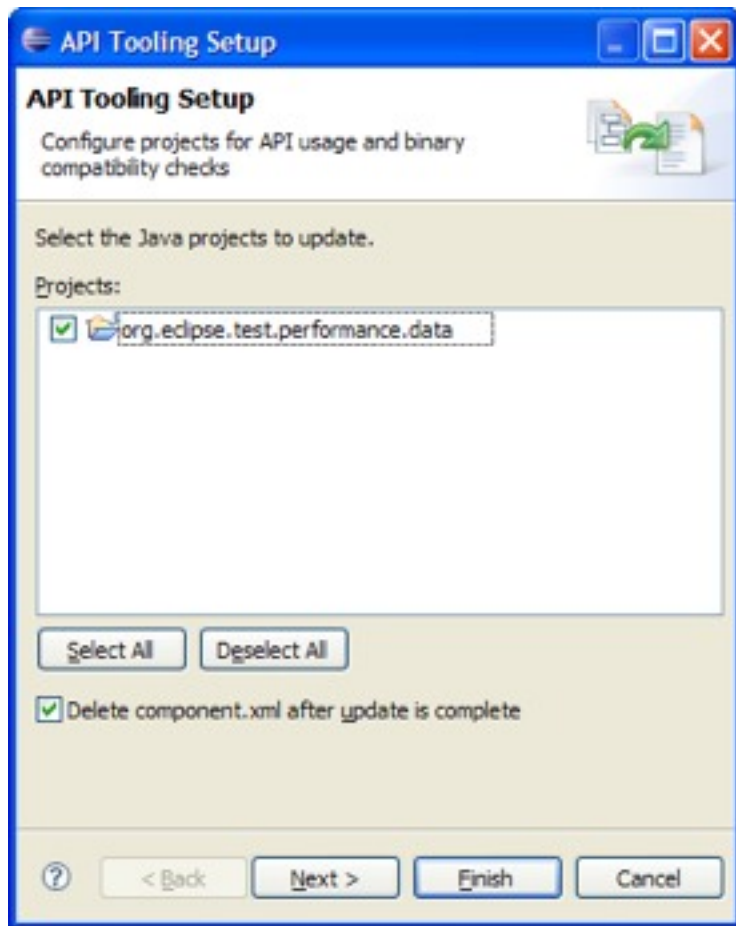
Build support

- 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)

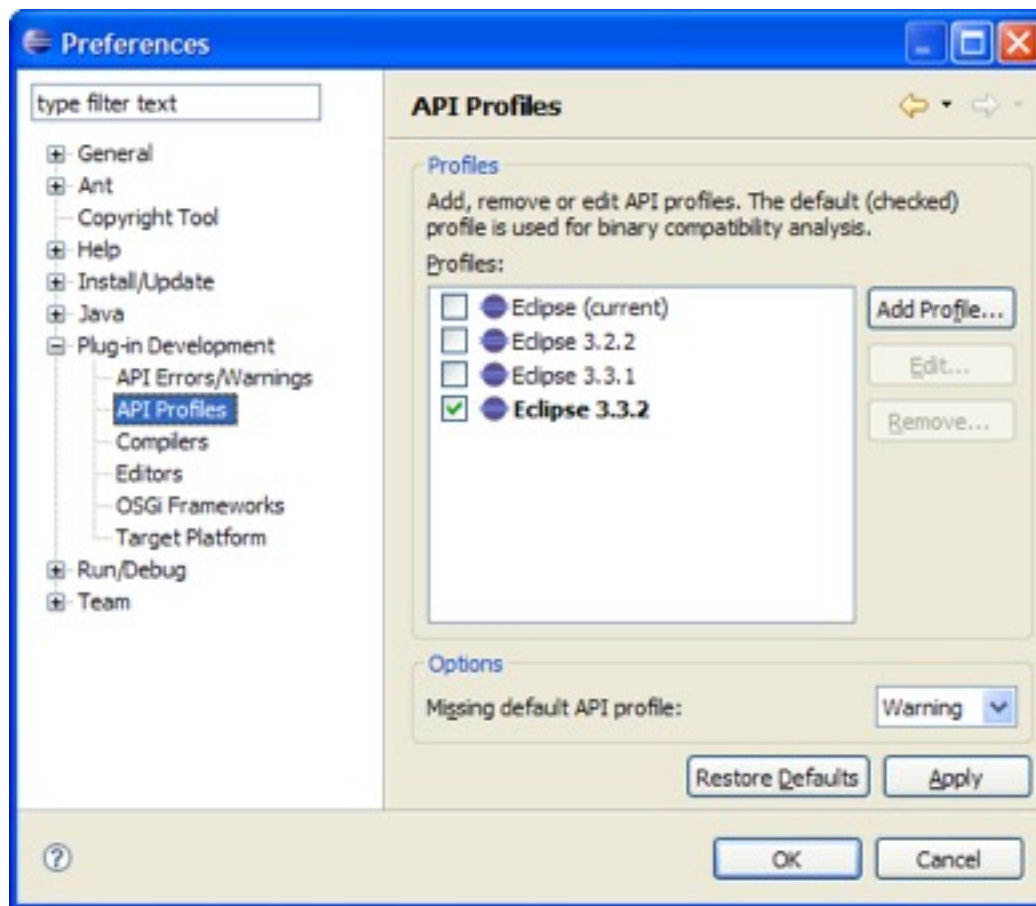
API Error/Warning Preferences



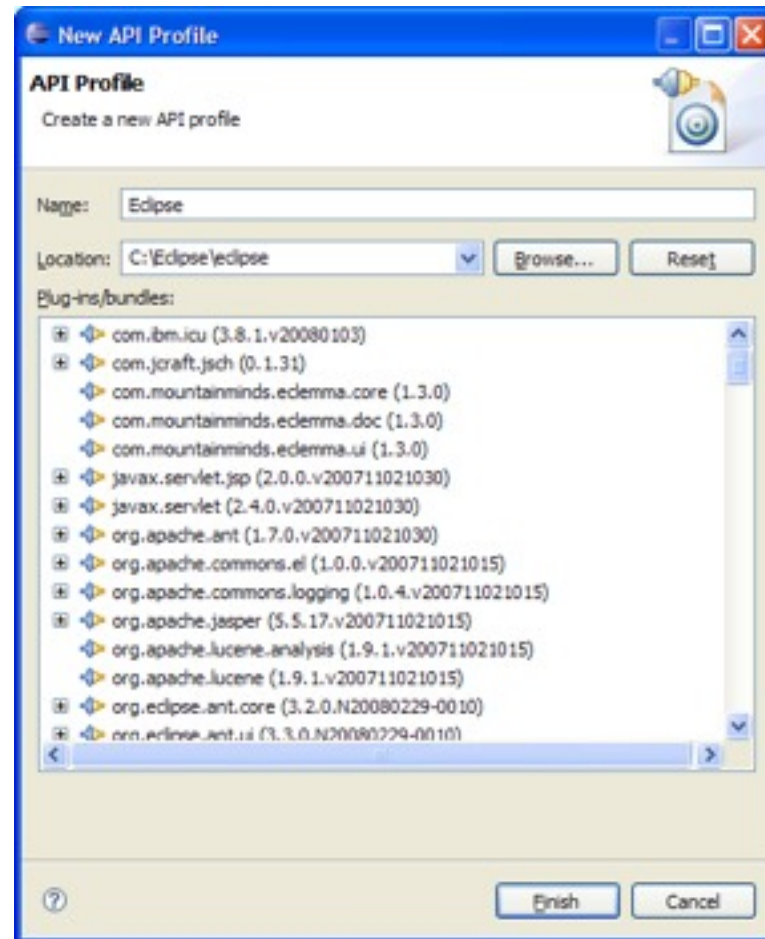
API Setup Wizard



API Profile Preferences



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

- Bugzilla

- ◆ https://bugs.eclipse.org/bugs/enter_bug.cgi?product=PDE



Q & A