PDE State of the Union

Chris Aniszczyk (EclipseSource)
zx@eclipsesource.com

Andrew Niefer (IBM)
aniefer@ca.ibm.com

Darin Wright (IBM)
Darin_Wright@ca.ibm.com

March 25th, 2009

Agenda

- API Tools
- UI
- Build
- The Future

Agenda

- **API Tools**
- UI
- Build
- The Future

Ideal Release Equation

Developer: ***Compatibility*** (producer)

+

Client: ***Honor API contract*** (consumer)

=

Free migration

The Ideal Never Happens...

Developer: (***features*** + ***deprecation*** +
Optional extensions to existing features +
Replacement API +
Provisional API)

+

Client: **Illegal internal references**

=

Migration at a cost

API Compatibility

- Assist developer with mechanics of API evolution
  - developers document APIs with restrictions (Javadoc tags)
- API Tooling identifies:
  - binary compatibility issues
  - breaking API changes, API changes after freeze
  - API leaks
- Suggests bundle version numbers
- Inserts and validates `@since` tags

API Consumption

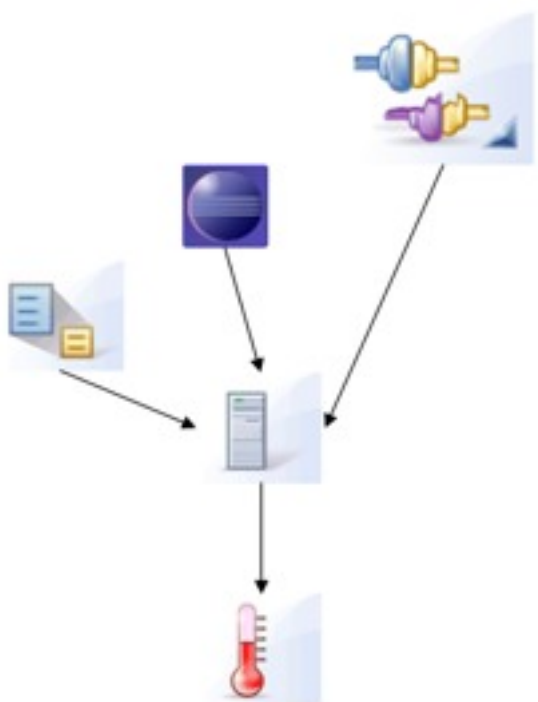
- API descriptions are shipped with bundles
- Assist client's task of honoring API contracts
  - tell them when they break the rules of engagement

Supported Restriction Tags

	Class	Interface	Method	Constructor	Final Field	Non-Final Field
@noimplement	-	✔	-	-	-	-
@noextend	✔	✔	-	-	-	-
@noinstantiate	✔	-	-	-	-	-
@nooverride	-	-	✔	-	-	-
@noreference	-	-	✔	✔	-	✔

API Tools in Build and IDE

- Analysis engine
  - Runs without OSGi or a workspace
  - Runs in engineering build (via ant tasks)
  - Runs in workspace (via builder)
- Inputs
  - bundles
  - baseline
  - problem filters
- Outputs
  - problems



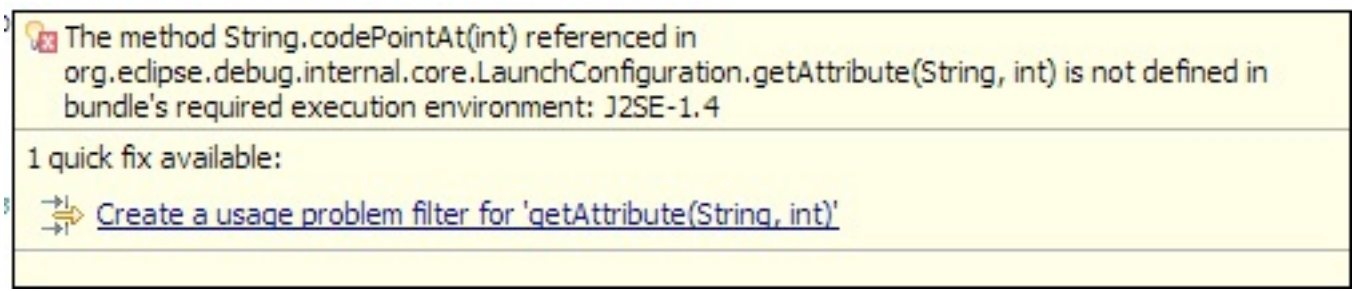
Infrastructure Enhancements

- Test Suite
  - Built a large regression test suite (>2000 tests)
  - Found and fixed a lot of bugs (>200)
  - Developed performance tests
- Improved
  - Analysis of split bundles (compare > compare.core)
  - API leak analysis (reduced false positives)
  - Performance of incremental build (in progress)

System Library Validation

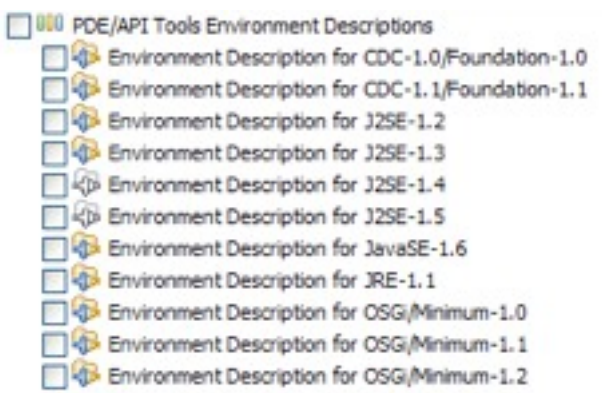
Validate system library use based on a bundle's required execution environment (BREE)

- if a bundle claims to run on J2SE-1.4, access to members in J2SE-1.5 and higher are flagged as illegal



API tooling provides descriptions

- install description fragments
- J2SE-1.5 description is 1.35 MB



API Usage Scans

- A *producer centric* view of APIs used by components
  - Find out which parts of your API are consumed by others
  - Set up scans for cross sections. E.g. what are the references between WTP and the SDK?

Bundle	API References	Internal References	Internal-Permissible References	Fragment-Permissible References	Other References
org.eclipse.ant.core	379	0	62	0	0
org.eclipse.ant.ui	0	2	0	0	0
org.eclipse.compare	2201	2	0	0	0
org.eclipse.compare.core	306	314	0	0	0
org.eclipse.core.commands	3757	0	0	0	0
org.eclipse.jdt.apt.core	27	0	165	0	0
org.eclipse.jdt.compiler.apt	0	0	22	0	0
org.eclipse.jdt.core	76143	38	17	2580	0

Who is {ab}using JDT?

Bundle	API References	Internal References	Internal-Permissible References	Fragment-Permissible References	Other References
org.eclipse.ant.ui	380	0	0	0	0
org.eclipse.jdt.apt.core	2063	0	0	0	0
org.eclipse.jdt.apt.pluggable.core	15	0	17	0	0
org.eclipse.jdt.apt.ui	11	0	0	0	0
org.eclipse.jdt.compiler.apt	58	0	0	2312	0
org.eclipse.jdt.compiler.tool	8	0	0	268	0
org.eclipse.jdt.core.manipulation	187	0	0	0	0
org.eclipse.jdt.debug	2239	3	0	0	0
org.eclipse.jdt.debug.ui	1735	0	0	0	0
org.eclipse.jdt.junit	1432	0	0	0	0
org.eclipse.jdt.launching	766	0	0	0	0
org.eclipse.jdt.ui	63870	0	0	0	0
org.eclipse.pde.api.tools	1390	32	0	0	0
org.eclipse.pde.api.tools.ui	349	0	0	0	0
org.eclipse.pde.core	497	0	0	0	0
org.eclipse.pde.ds.core	6	0	0	0	0
org.eclipse.pde.ds.ui	100	3	0	0	0
org.eclipse.pde.runtime	5	0	0	0	0
org.eclipse.pde.ui	1032	0	0	0	0

Why API Usage Scans?

API producers can see what internals are being used

- Inform clients how to use proper API
- Determine what new API needs to be added
- Know what clients will be broken
- Avoid breaking internals until clients have migrated

Future Directions?

- Consumers provide producers with API use scan
- Tooling warns when producer breaks a dependency
- Central registry of API use scans

Agenda

- API Tools
- **UI**
- Build
- The Future

Target Definitions

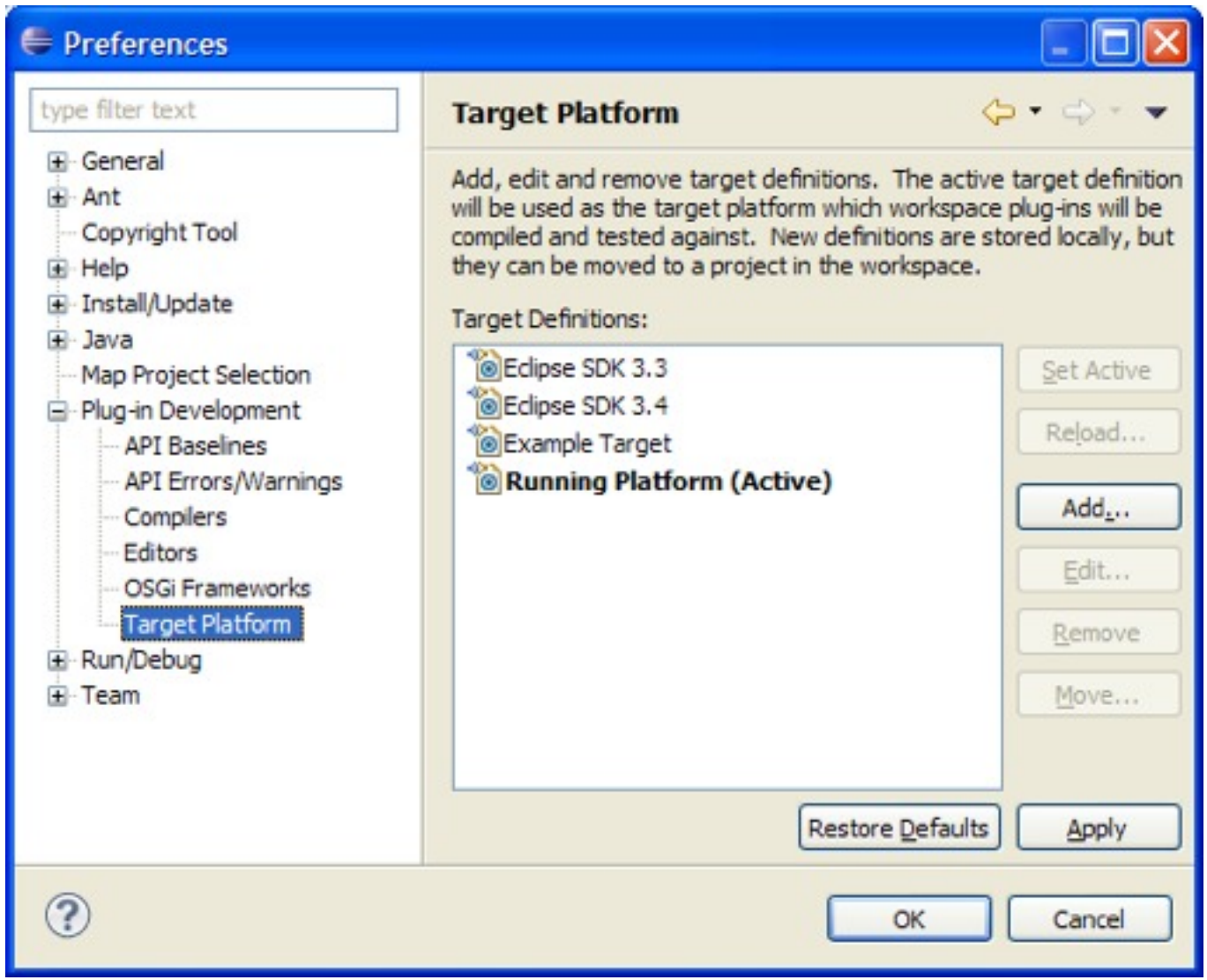
A fundamental task of PDE is to manage the set of plug-ins (bundles) that the workspace is compiled and run against

- Provisional API for target definitions
- Compose target with various bundle sources
  - installations vs. directories
  - features
  - set of installable units from an update sites/p2 repositories
- Leveraging the API within PDE
  - an API baseline is just a target definition
  - plug-in import

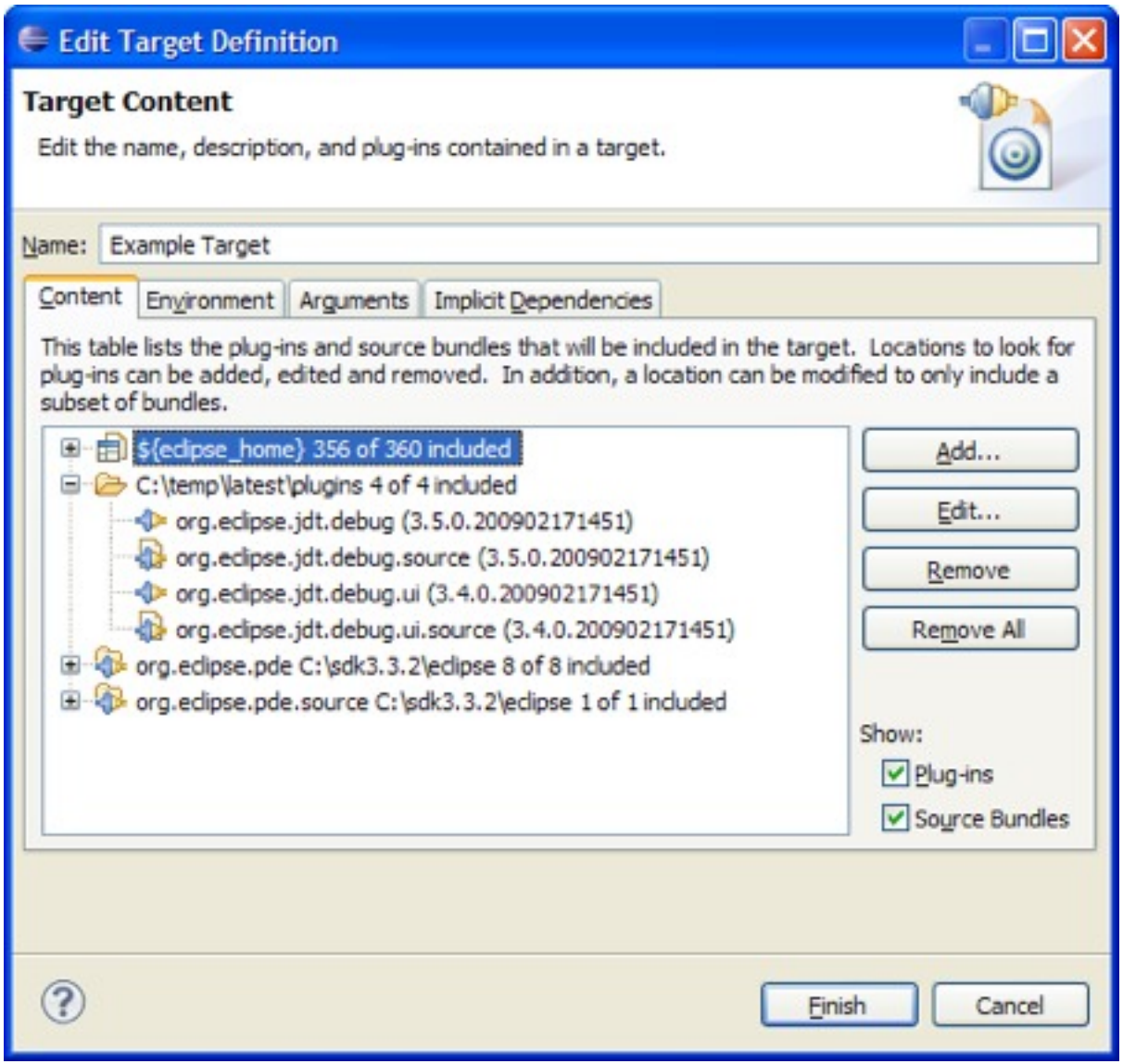
Target Definition User Interface

- Instead of having one target platform, the user can manage a set of target definitions and switch between them
- Targets definitions are backed by files in the metadata or workspace
- Preference page sets workspace's active target and provides wizards to create/edit targets
- Users can modify target definitions using an editor

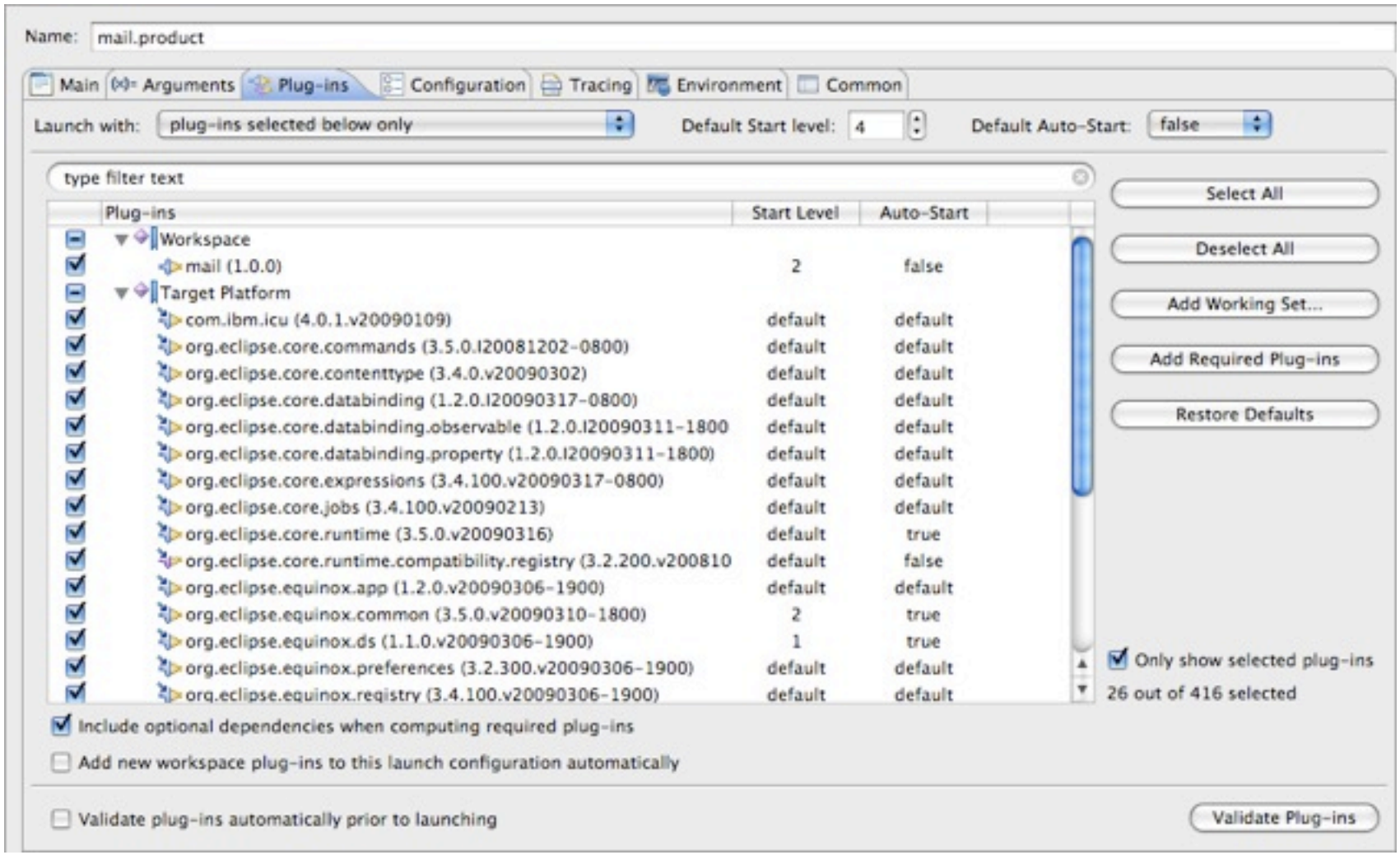
Manage Target Definitions



Edit Target Definitions

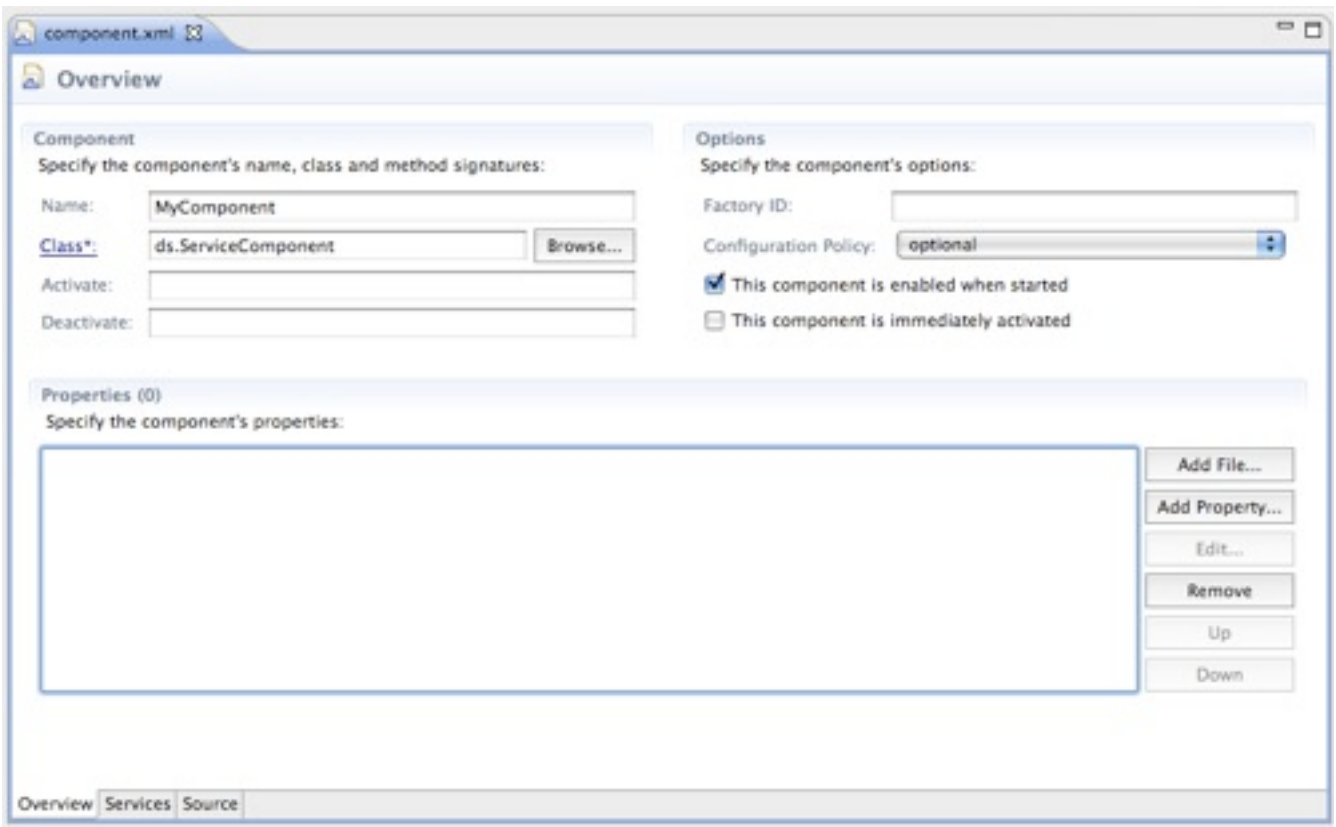


Start Levels



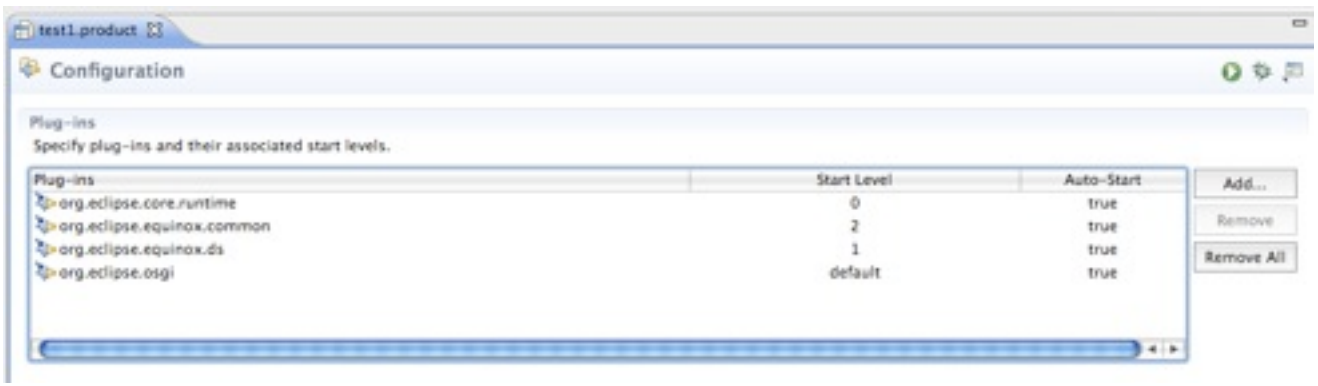
Declarative Services Tooling

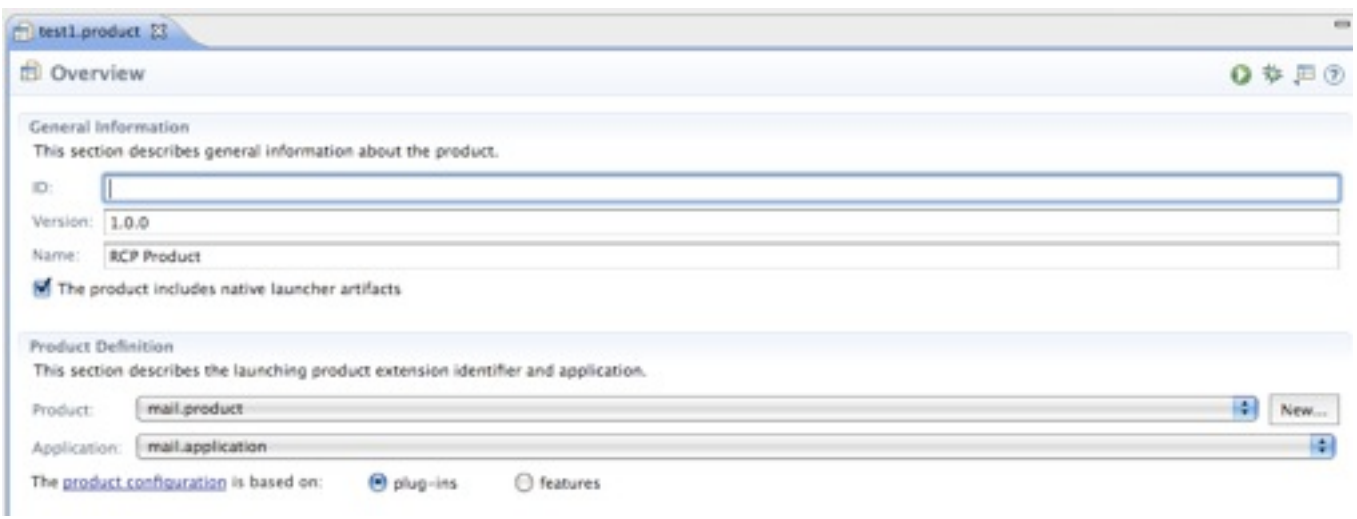
- OSGi Declarative Services in Eclipse 3.5
- PDE provides DS Tooling



Product Definition Enhancements

- Versions
- License
- Start levels
- Properties
- Optional Product Extension





Plug-in Export Enhancements

Export and install into running platform

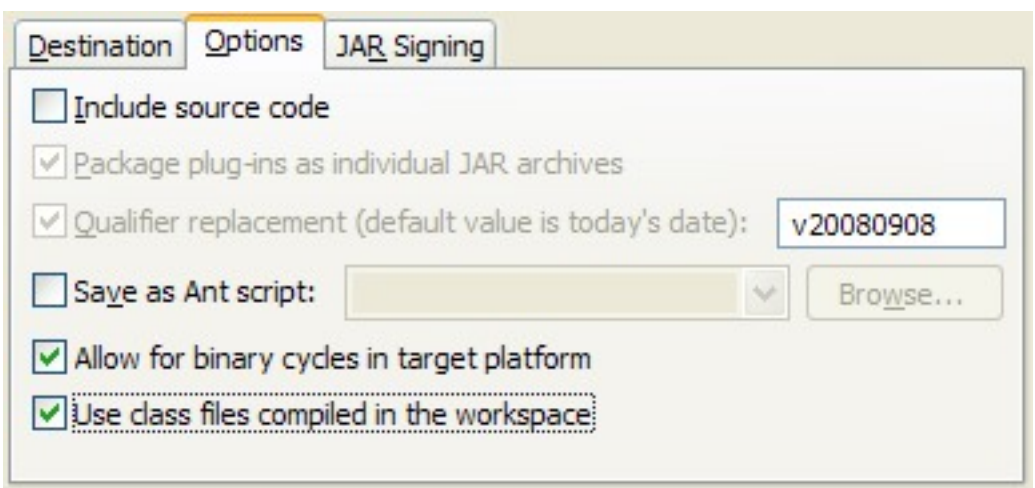
- build and export plug-ins and install them into the running host
- replaces the old "export, copy, paste, delete, restart" slam technique

Export existing class files from workspace

- avoid re-compiling

Generate source bundles

Keypass support



Plug-in Export and p2 Metadata

Export with metadata

- Current support uses 3.4 metadata generator (binary jars)
- Coming in 3.5 metadata support using p2 publisher (source)

Exporting Products

- Coming in 3.5, product export with metadata will perform a director install to get a fully p2 enabled product.

Agenda

- API Tools
- UI
- **Build**
- The Future

Build and p2

Building with p2: "`p2.gathering=true`"

- Supports publishing from source by gathering compile results
- Feature builds result in a p2 repository
- Product builds result in a installed p2 enabled product (or repo)
- Consume zips of repositories in your build via transformed repos

p2 releng tools

- Process artifact repositories
- Sign, perform pack200 processing
- Mirror tools:
  - Mirror using a baseline to keep pre-existing binaries.
  - Use a comparator to detect differences against pre-existing binaries.

Sorting Dependencies

In Eclipse 3.4 and earlier, compilation is done via the feature hierarchy

- Features are visited depth-first.
- Within a given feature, plug-ins are compiled in dependency order
- Required careful management of feature hierarchy

In Eclipse 3.5, we can sort bundles according to dependencies

- `parallelCompilation=true`
- `flattenDependencies=true`

```
<project name="Compile master" default="main">
   <target name="main">
      <parallel threadsPerProcessor='3'>
         <ant antfile="build.xml" dir="plugins/org.junit4" target="build.jars"/>
         <ant antfile="build.xml" dir="plugins/org.eclipse.rcp" target="build.jars"/>
         <ant antfile="build.xml" dir="plugins/org.eclipse.osgi" target="build.jars"/>
      </parallel>
      <parallel threadsPerProcessor='3'>
         <ant antfile="build.xml" dir="plugins/org.eclipse.jdt.junit4.runtime" target="build.jars"/>
         <ant antfile="build.xml" dir="plugins/org.eclipse.jdt.doc.user" target="build.jars"/>
         <ant antfile="build.xml" dir="plugins/org.eclipse.jdt.doc.isv" target="build.jars"/>
      </parallel>
      ....
```


More Build Improvements

- Performance Improvements
  - reusing .class files from the workspace
  - avoiding some copying while building plug-ins
  - publishing to a p2 repository directly from source
- Signing with Keypass
- Per Feature Individual Source Bundles
- Product build improvements
- Custom Execution Environments

Agenda

- API Tools
- UI
- Build
- **The Future**

API Tooling

- API comparison views/reports
  - show me the API changes between two releases
- Determine compatible version ranges
  - of require bundles
  - as dependencies are introduced/removed
  - which versions satisfy my set of references
- Analyze pure Java projects
  - need a way to specify package visibility (API descriptions)
  - JSR 277 - Java Module System
- Tighter integration with Java builder
  - feed a fine grained set of access rules to the Java compiler
- Package level versioning
  - agree on a versioning specification
- Analyze extension points & extensions

Target Management

Support target platform per project in the workspace

- share target definition on classpath with team
- share API baseline on a classpath with team

API to compose/manipulate targets

Support provisioned targets via repositories

- bundles are actually installed into target (using p2)
- has a managed profile
- running/debugging installs workspace bundles and leverages OSGi framework to perform launch

Unify target concepts/implementation

- target platform, launch configs, products all have similar information

Launching

Support other OSGi Frameworks

Take advantage of new OSGi Launching APIs

Build

- Exporting p2 from the UI
- Further Integration with p2
  - Improved fetching from p2 repositories
  - Improved Reuse of p2 metadata
  - Publishing p2 products
  - Repository management tools

The perennial favorites:

- *Incremental building ?*
- *Workspace integration ?*

OSGi and Community Building

We aren't alone in the bundle tooling world!

No reason to reinvent the wheel!

PDE is attending the OSGi Tool Summit on Friday

Thank you for your attention!

- Questions and feedback welcome!

Chris Aniszczyk (EclipseSource)
zx@eclipsesource.com

Andrew Niefer (IBM)
aniefer@ca.ibm.com

Darin Wright (IBM)
darin_wright@ca.ibm.com

