

Share this plug-in with the ones you love: Using PDE to create an Eclipse plug-in and publish it on the update site

Dr Dejan Glozic,
Manager, Eclipse Components
IBM Canada Ltd., Toronto Laboratory

Just unzip...

- A way cool method of sharing plug-ins – unzip into 'plugins' directory
- Developers can exchange plug-ins via email
- Eclipse discovers plug-ins on each startup



Not as cool?

- Unzipping is way too 'hackerish' for any self-respecting product
- Error-prone – various ways of shooting yourself in the foot (contest: find your 23 plug-ins in the hay stack of 700)
- It is easy to unzip – it is the un-unzip that bites
- It is easy to unzip once – unzipping for the 10th time gets somewhat boring
- In this day and age when we put the man on the Moon, can't we do better than use unzip?



Features – the best kept Eclipse secret

- There is a better way – group your plug-ins into **features**
- Publish features into **update sites** on an HTTP server; save email for sharing the URL
- Let **Update Manager** locate, download and install your features into the Eclipse product

Advantages

- If something goes wrong, users can **revert** to the last good configuration (without your Trojan horses; bad, bad developer!)
- After that, users can **uninstall** your features into oblivion
- On the other hand, users can **search for updates** of good features or configure Update to do it **automatically**
- You can publish **patches** for your bug-free features



Anatomy of a feature

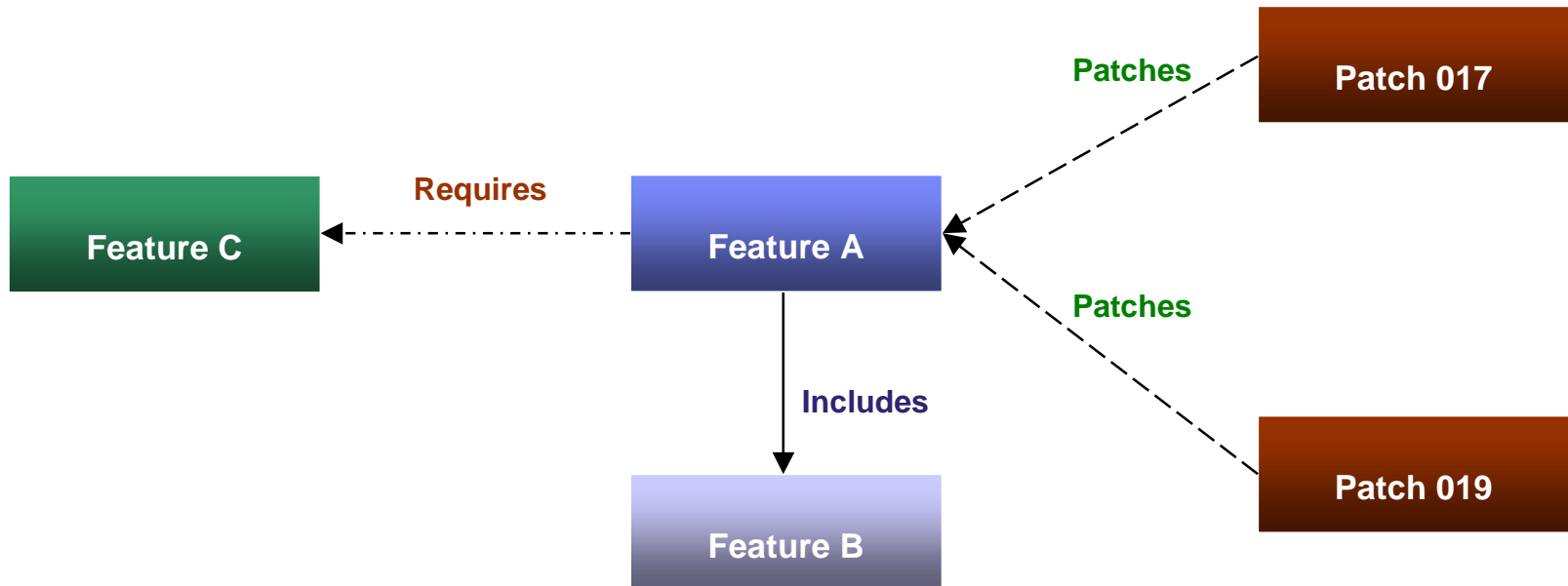
- It has a name, unique ID, version, provider
- It groups 0 or more plug-ins and 0 or more non-plug-in archives
- Provides for defining license, copyright, short and long description

Update site references

- One update site URL – will be used to search for updates
- Zero or more ‘discovery’ URLs – allows one feature to bring in URLs for other interesting update sites



Feature relationships



Feature Manifest

```
<feature id="acme" version="1.0.0" provider-name="Acme Inc."
        label="A Cool Acme Feature">
  <url>
    <update url="http://acme.com/updates/" />
    <discovery url="http://acme.com/utils/" />
  </url>

  <description url="moreinfo.html">
    A simple description of a feature
  </description>

  <license url="license.html">
    License Agreement as text
  </license>
  <includes id="acme1" version="1.0.0" />
  <requires>
    <import feature="org.eclipse.platform" version="3.0.0" />
  </requires>

  <plugin id="com.acme.foo" version="1.0.0" />
</feature>
```

Bringing in the non-plug-in content

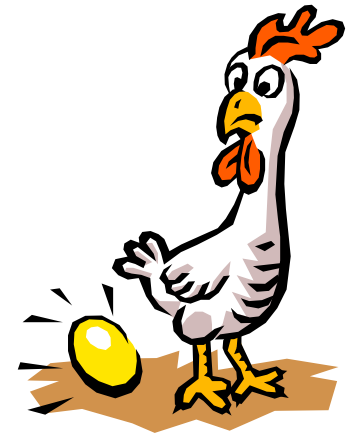
- Eclipse Update knows the semantics of plug-ins in features – it can install them without assistance
- You can add non-plug-in archives (JARs) that package random resources
- Update will download the archives but will not know what to do with them

Enter install handlers

- Java classes that implement handler interface can be associated with features
- Install handler life cycle methods are called for all install operations (before and after)
- Custom install handler can take the random JAR and do something with it

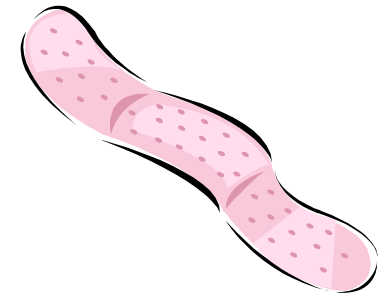
Baseline features and updates

- A classic chicken-and-egg dilemma: you cannot install Eclipse platform feature from an update site because you need platform feature to run Install wizard and browse the site.
- Eclipse products come with plug-ins and features in the file system – this is the baseline content
- The features contain embedded update URLs that will be used to search for updates
- When updates are found and installed, old features are not deleted – just unconfigured (disabled)
- Several versions of the same feature coexist on the file system – only one is ‘visible’ to the runtime



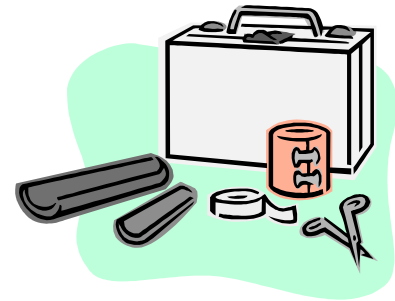
Patches

- Updates require that features are replaced with those of the same ID but a higher version
- For large products, it is hard to ship 'true' updates every time an emergency fix is needed
- Patches live side-by-side features they patch – they just bring new versions of select plug-ins (3.0 behavior)
- Eclipse run-time sorts things out – picks the newer (patched) plug-ins



Patch Properties

- Feature can be defined as a patch for a particular version of another feature
- When searching for updates, both updates and patches are reported
- When a feature is disabled, its patches go with it
- When a feature is uninstalled, its patches are uninstalled as well



Conditional entries

- Plug-in entries can be made conditional to os/ws/arch values
- They will only be taken into account if os/ws/arch combination matches
- Example - Eclipse Platform feature:

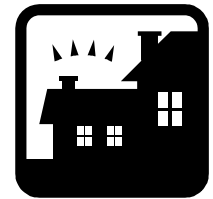
```
<plugin id="org.eclipse.swt.gtk" version="3.0.0"  
  fragment="true" os="linux" ws="gtk" />
```

```
<plugin id="org.eclipse.swt.motif" version="3.0.0"  
  fragment="true" os="linux" ws="motif" />
```

```
<plugin id="org.eclipse.swt.win32" version="3.0.0"  
  fragment="true" ws="win32" />
```

Product Extensions

- Several products built on the common stack may want to share content
- Features and plug-ins of a product may be grouped into the **product** location and one or more **extensions**
- Each location has its own **plugins** and **features** directories
- When such a product is installed while another is already present, it can extend the existing product
- Extended product may have read only access to the extension, or it can be allowed to update and/or patch it
- Using Help>Software Updates>Manage Configuration..., extensions can be disabled as a while



Automatic Updates (Eclipse 3.0)

- Configure Eclipse to search for new updates
- The search can be on each startup, or periodically (e.g. every Monday at noon)
- The search is executed using Eclipse 3.0 background job scheduler
- Users are informed if something new is found
- If configured, Eclipse Update will go ahead and download the new updates in the background, then ask if it is OK to install



Update Site Mirroring (Eclipse 3.0)

- Needed for large shops with many product seats
- Local administrator mirrors the product update site to the local server behind the firewall
- All products are configured to use the update policy on that server (a policy URL set in the preferences).
- Automatic or manual search is redirected to look in the local site instead of connecting to the product site on the Internet
- Combination of the local mirror and automatic updates allows administrators to effectively 'push' updates to clients



Shared Installation Support (Eclipse 3.0)

- When Eclipse is installed on a shared volume/drive, local configuration is cascaded to the shared configuration
- New features or updates in the shared installation are automatically picked up
- Features installed locally by the client are managed by the client
- It is possible to 'freeze' shared configuration – it becomes local and changes in the shared install do not affect clients

May appear in Eclipse 3.0

Feature sets

- customized lists of features from the server that the client 'sees'
- provides for subset of the long list of features in the shared install
- Features listed using their names – no versions
- Implemented as filtering on top of the configuration
- If the user wants 'Web Tooling', he will get the currently configured web tooling features (say, version 5.0.0)
- If the administrator upgrades web tooling, the user will receive the upgraded version (say, 5.0.2).

Other Eclipse 3.0 improvements

Sub-plug-in patches

- It is possible to prepare a patch that only carries files that changed
- Eclipse Update locates the patched plug-in, makes a copy, then overlays it with the new files
- Faster to install – less bytes to download across the network

Restartable downloads

- If failed during a large file download, Update can pick up where it left off if restarted
- Only works within the same session (must not restart Eclipse)

Other Eclipse 3.0 improvements (cont'd)

Installation from a zip

- The entire update site can be in a zip file
- Update is given the URL of the file in the file system or CD-ROM

Command Line Updates

- All update operations available on the command line via a headless application
- Content of the remote update site can be listed on the console
- One or more of the listed features can be installed
- Existing features in the configuration can be enabled, disabled and/or uninstalled.

Demo

- Create 'Hello, World' plug-in
- Create a feature
- Create an update site
- Build and publish the feature
- Install the feature into another Eclipse
- Verify that we have it
- Create the new version of the plug-in/feature (an update)
- Publish it
- Start another Eclipse and let automatic updates pick the feature up
- Verify that we have the new version
- Thank the audience for the applause



Useful Links

- Online help for Install/Update in Eclipse 2.1:
<http://help.eclipse.org/help21/index.jsp?topic=/org.eclipse.platform.doc.user/tasks/tasks-36.htm>
- Install/Update component home page:
<http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/platform-update-home/main.html>
- PDE component home page:
<http://dev.eclipse.org/viewcvs/index.cgi/%7Echeckout%7E/pde-ui-home/main.html>
- 'How to keep up to date' article on Eclipse.org:
<http://eclipse.org/articles/Article-Update/keeping-up-to-date.html>