Le fichier bundles.info

Les mécanismes pour supporter les plug-ins sont implémentés en utilisant le framework OSGi.
un plugin = un bundle OSGi
Le fichier **bundles.info** contient une liste de tous les plug-ins installés dans le système courant. Au démarrage d'Eclipse, tous les plug-ins listés dans ce fichier sont donnés à OSGi comme l'ensemble exact de plug-ins avec lequel s'exécuter.

La syntaxe acceptée pour le listing des bundles est :
`<bsn>,<version>,<jar-location>,<start-level>,<toStart?>`
Avec :
bsn - Le nom symbolique du bundle
version - La version du bundle
jar-location - Chemin relatif ou absolu vers le fichier jar
start-level - Un chiffre indiquant le niveau de démarrage du bundle
toStart? - Une valeur true ou false indiquant si un bundle devrait être démarré ou non

Exemple :
org.eclipse.virgo.util.osgi,3.1.0.BUILD-20111031165127,plugins/org.eclipse.virgo.util.osgi_3.1.0.BUILD-20111031165127.jar,4,true

Plugin fragments

Il est parfois utile de rendre une partie d'un plugin optionnelle, lui permettant ainsi d'être installée, désinstallée ou mise à jour de manière indépendante par rapport au reste du plug-in.
Un plug-in peut avoir par exemple une bibliothèque qui est spécifique à un OS particulier.
Dans cette situation, vous pouvez créer un fragment qui est associé à un plug-in hôte particulier. Sur le disque dur, un fragment ressemble presque trait pour trait à un plug-in, excepté quelques différences esthétiques :

• Le manifest est stocké dans un fichier appelé `fragment.xml` au lieu de `plugin.xml`
• L'élément de niveau le plus haut dans le manifest est appelé fragment et a deux attributs supplémentaires – `plugin-id` et `plugin-version` – pour spécifier l'ID et le numéro de version du plug-in hôte.
• Le manifest du fragment n'a pas besoin de son propre élément `requires`. Le fragment va hériter automatiquement de l'élément `requires` de son plug-in hôte.

Il peut ajouter des éléments `requires` s'il a besoin d'accéder aux plug-ins qui ne sont pas requis par le plug-in hôte.

Mises à part ces différences, un fragment ressemble à un plug-in normal. Un fragment peut spécifier des librairies, des extensions, et d'autres fichiers. Quand il est chargé par le loader de la plateforme, un fragment est mergé de manière logique (et non physique) dans le plug-in hôte. Le résultat final est exactement le même que si le manifest du fragment était copié dans le manifest du plugin, et tous les fichiers dans le répertoire du fragment apparaissent comme s'ils étaient situés dans le répertoire d'installation du plug-in.

Optional dependencies

Dans certaines situations, votre code peut marcher, qu'un bundle spécifique soit disponible ou non - il peut être capable de s'ajuster dynamiquement à cette condition (par exemple, utiliser les services exportés s'ils sont là, ou suivre un chemin alternatif dans le cas contraire). Vous pouvez spécifier une telle dépendance optionnelle en marquant le bundle requis comme « optional ». Quand cette option est activée, cela indique à la runtime que votre plug-in devrait s'exécuter même si une dépendance de plug-in optionnelle est manquante ou non résolue.

Liens

http://eclipsesource.com/blogs/2009/04/29/plug-in-bundle-blugin/

http://help.eclipse.org/luna/index.jsp?topic=
%2Forg.eclipse.platform.doc.isv%2Fguide
%2Fruntime_model_bundles.htm

https://wiki.eclipse.org/FAQ_What_is_a_plug-in_fragment%3F

http://www.developer.com/java/ent/article.php/3655231/Eclipse-Tip-Use-Optional-Plug-in-Dependencies-to-Support-Diverse-Runtime-Environments.htm

http://help.eclipse.org/luna/index.jsp?topic=%2Forg.eclipse.pde.doc.user%2Fguide%2Ftools%2Feditors%2Fmanifest_editor%2Fdependencies.htm