1 Understanding TextGridLab Updates

This article is work in progress

TextGridLab uses Equinox p2 for its update mechanism. p2 has the notion of **Installable Units (IU)** that can be installed from a list of repositories to a target profile – these installable units correspond to Eclipse features, plugins and products, the target profile is basically the installed and configured application, and the repositories are the update sites. Installable units have an identifier and a version, and they can have dependencies on other IUs within specific version ranges.

When the user wishes to install or update an IU, p2 tries to draft a *provisioning plan*, i.e. it works out what needs to be installed or updated such that the requested IU(s) are installed to the target profile and all dependencies are satisfied.

1.1 Automatic updates

It is important to note that the automatic updates (cf. Updating in the User's manual) do not check for updates of all IUs installed in the application, but only for updates to **root IUs**. Root IUs are only those that have been installed explicitely to the target profile, not those that have only been drawn in by dependencies.

1.1.1 Listing root IUs

All root IUs are installed to a specific TextGridLab installation are listed as top-level items on the *Installed Software* tab in the dialog that can be summoned using *Help About TextGridLab Installation Details …*. They can also be listed at the command line by using the director application (on windows, use the bundled eclipsec.exe intead of textgridlab.exe for command line execution):

```
…/TextGridLab % ./textgridlab -nosplash -application org.eclipse.equinox.p2.director -lir
info.textgrid.lab.base.product/2.0.90.201210021355
```

1.1.2 Root IUs in TextGridLab installations (at least 2.0.x)

As you can see, in a vanilla TextGridLab 2.0.1 installation, there is only one root IU, and that corresponds to the product. If you expand the tree in the dialog, you'll see all other components somewhere in the root product's dependencies.

If you use the Marketplace to install more tools you will see those tools listed as root IUs (and hence update-checked), as well.

However, if we update, e.g., (only) the XML editor bundles on the stable update site, you wouldn't be notified of the update unless we'd also update the product *and force a version update through the product specification*. You can, of course, manually install a new version using the software installation dialog—this will also turn the new version of the updated feature to a root IU that will be explicitely checked for updates from now on. But from the developer's point of view, this is not a way to push new software to the users.

1.1.3 Dependencies and version ranges generated from Eclipse specification files

So what is updated additionally to the root IUs? That is determined by the (direct and transitive) dependencies of the root IU. It is important to note that a dependency from A to B is typically specified with a *version range*, that every version of B that falls into the version range specified for the dependency on B by A satisfies this dependency, and that p2 will not update B to a newer version if the installed version of B satisfies all dependencies (and is not a root IU). It should also be noted that Eclipse features are metadata collections that merely manage a set of dependencies instead of directly containing the plugins they consist of. So where can the dependencies and version ranges that are followed by p2 can be specified?

1.1.3.1 Product

The **product specification** (i.e. the `.product` file) contains in TextGridLab's case a list of features without version number. At build time, these dependencies are augmented with the absolute version numbers that are also used to assemble the product ZIP—i.e. there are no version ranges here, the product depends on specific versions of its included features.

1.1.3.2 Features

Features offer various mechanisms for expressing dependencies:

- **feature (and plugin) dependencies** go to features with version ranges. Those version ranges are not specified as intervals like in p2 metadata, but by a version number and a match rule (compatibility level).
- **feature inclusions** go to a specific version of a feature. Developers can specify the special version number `0.0.0` which will be replaced with the specific version of the feature that is chosen at build time.
- **plugin inclusions**—i.e. the list of plugins the feature consists of—work in the same way as feature inclusions, i.e. you should specify `0.0.0` and the specific version number will be inserted at build time

1.1.3.3 Caveats

- Updates of non-root IUs will only happen if they are enforced by root IUs.
  Example:
  - The *product* includes the *core* feature (specific version).
  - The *core* feature depends on the *tgcrudclient* feature (version range).
  - The *tgcrudclient* feature is updated, the rest remains untouched.

  Now existing product installations *will not* update to the new *tgcrudclient* feature since only the product is checked for updates.

1.1.4 Generating new root IUs

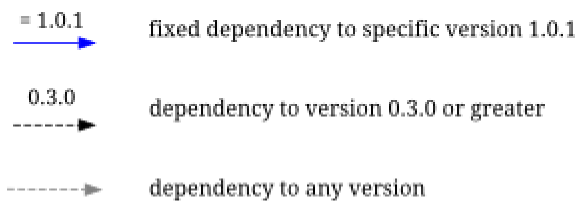
It is theoretically possible to create a product that has some of its features installed as root IUs, cf. Andrew Niefers blog post and the corresponding Tycho bug. Requires

- a <product>.p2.inf specifying dependencies to the root features & their version ranges
- a p2 director call in the final assembly stage to actually install this stuff. Seems be supported in the upcoming Tycho version 0.16 (not tested yet)

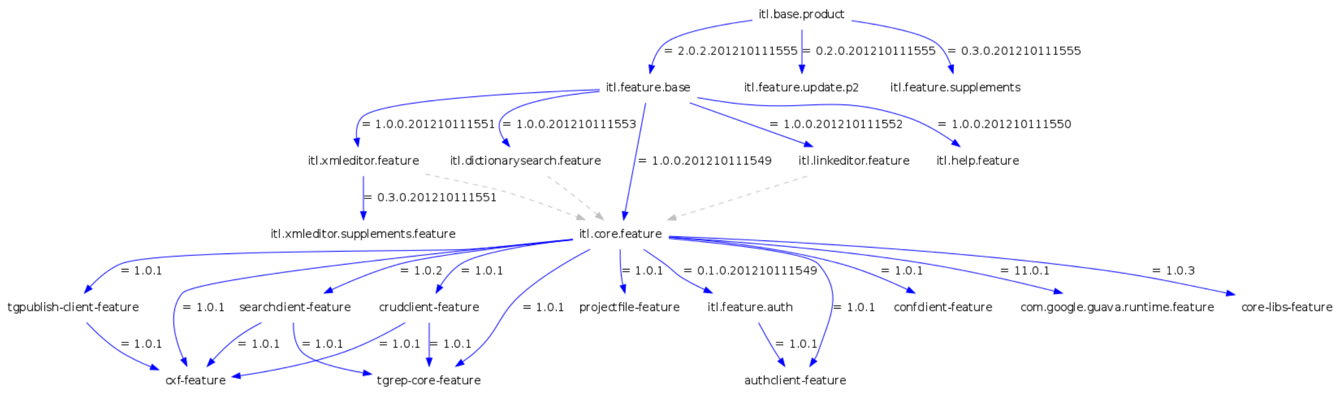
1.1.5 Feature version dependencies in TextGridLab

I've created some graphs illustrating feature dependencies in TextGridLab.

> The dependency graphs have been generated from the repository of the *base* projects. As these are the source for the generation of the TextGridLab products, they contain everything required. The graphs plot only products and features. Additionally, everything starting with `org.eclipse` has been filtered to avoid cluttering the graph with stuff we're not interested in, anyway. The key to the graph is shown below. Dependencies (dashed arrows) can also specify OSGi version ranges, e.g., `[1.1.0,2.0.0)` means all versions starting from 1.1.0 up to, but not including, 2.0.0.

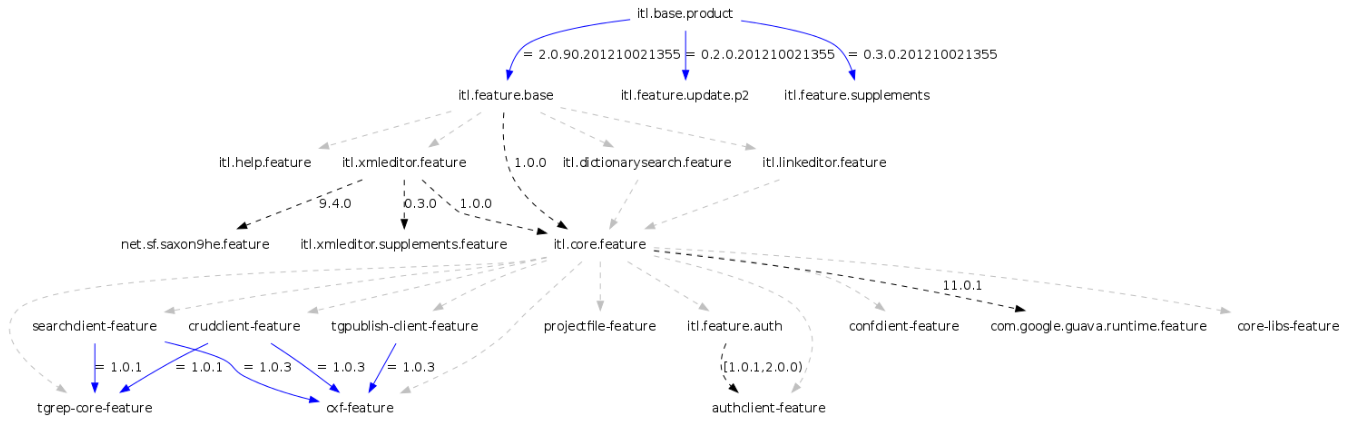

= 1.0.1
fixed dependency to specific version 1.0.1
0.3.0
dependency to version 0.3.0 or greater
dependency to any version

First graph illustrates version dependencies in the TextGridLab 2.0 line, here it's from the candidate for TextGridLab 2.0.2:



As can be seen, *every* feature is referenced using at least one fixed dependency, with the consequence that everything will be updated on an update.

For the current nightly branch, the graph looks like this:



There are fixed dependencies between the features from the *dependencies* project (at the bottom of the image), but between many of the features, there are unversioned dependencies—with the consequence that no update will be enforced here.