- Getting list of F27 packages#

Data Provider#

- The data provider uses a package definition file hosted on F27 to get the list of packages

Package definition file#

- Similar to a memento. It contains packages details including:
  - Title - name of the package
  - Release - package version
  - CPU - target CPU variant
  - Source dependency - Reference to the dependent packages
  - Containing projects - Reference to
  - Vendor - package provider (mainly QNX)
  - Repository - The F27 repository (e.g. http://community.qnx.com/svn/repos/coreos_pub)
  - Location - the F27 hosting URL within the repository (e.g. trunk/services/system)
  - Description - the package detailed description
- The definition file will be maintained by F27 admin (?)
- The file is publicly accessible, can be downloaded without authentication. Location of the file is - TBD
- The IDE does not cache any package information locally but always queries the definition file at the beginning of every new import section

How packages are defined?#

- What is the granularity?
- Source or binary format?
- Make sure packages are build-able before posting?

F27 login credential - two scenarios#

A per-repository login prompt#

- Similar to the update manager login panel
- User can have different username and password on different repositories (possible?)
- Username and password are recorded to "Team -> SVN -> Password Management" for future re-use

A preference page to save login username and password#

- Applicable when using the same login info for all repositories
- Has to encypt the password. Leverage the platform "Secure Storage" mechanism (?)
- Still has limitation - save in workspace only. Can it be exported to other workspace?

Package dependencies#

- Dependencies are defined in the "module.tmpl" <requires> tab, example for "trunk/services/system" is:

```
<!-- Dependencies -->
  <requires>
    <part build="false" location="lib/elf"/>
    <part build="true"  location="lib/c"/>
    <part build="false" location="hardware/startup/lib"/>
    <part build="false" location="utils/m/mkasmoff"/>
  </requires>
```

- To build the complete dependency, all the referenced packages' module.tmpl files have to be fetched locally and parsed
- At this phase, only module.tmpl files are fetched, not the whole package tree

- Presenting packages#

The model#

Refactor existing model#

- Abstract the existing "PackageContainer" model to a "globalPackageContainer"
- Refactor current source zip file model to be one implementation of the "globalPackageContainer"
- Create a new remote package model as another implementation of the "globalPackageContainer"
- Refactor all the model classes (BSP, sourcePackage) using the same manner

The remote package model#

- The remote package model is an abstract model by itself with different implementation for remote binary file and remote source tree
- Based on the contents of the package definition file and the package dependency hierarchy, the remote package model is built as a tree like structure
- Each tree node is represented by a "packageItem" class. It has properties like "title", "release", "cpu", "repositoryLocation", "dependingOn", "dependedBy", "enabled", etc. There are helper methods to get/set properties
- A "packageManager" class holds the model objects and is responsible to manage the tree nodes. It has methods like "add()", "remove()", "convertTo()", "enable()", etc.
- Interfaces will be used to defined APIs to the tree node and the manager so that they can be extended to different type of packages.

The U.I.#

- TBD

- Action#

The action is invoke at the end of the wizard when user click "Finish", not from any context menu or toolbar icon

Checking out packages#

- Depending on the selection in the U.I., the action does either a SVN "checkout" or "export"
- In both scenarios, the action preserves the source tree layout as the same as it's posted on F27
- For dependent packages, it prompts the user to check them out at the same time and gives "project may not build if select no" warning in the prompt

Creating projects#

- Projects to be created are defined by searching for "module.tmpl" files in the packages.
- Project references are defined by the project dependencies. When creating the projects references should be created as well
- Should leverage the existing source import action. It currently does an import on the source zip file, will extend it to support source tree checkout