01 - organize a plugin

Plugin structure

The plugin is composed of two main packages:

- ***_plugin_***
  Contains the configurations file that are used in order to expose the plugin services. This package is automatically scanned by the application distribution loader and the configurations files within are interpreted by using the ally IoC (Inversion of Controls) dependency injection container.
- **<main package>**
  The main package of the plugin, all the implementations need to be in this package. The name of the main package needs to be lower case only separated by underscores. This name needs to be as unique as possible among the other packages an needs to reflect the purpose of the plugin. Plugins that are parts of a bigger functionality and that will make sense to have the same main package name can also have sub packages to separate the functionality within the plugins group. Some examples from existing plugins: **introspection**, **sql_alchemy**, **superdesk**.

Beside this main packages the plugin also needs to contain a build file. We currently use ant builds in order to package the plugins, they are easy to use from eclipse.
The build file is called "**build-plugin.ant**" this build will need to package the plugin to an egg file to be used by the application distribution.

The plugin package "*_plugin_*"

Also the **plugin** *_package needs to have a defined structure. Since this package is automatically scanned and used within the application is very important to respect the structure. The **plugin** will not contain any modules beside the **init** module which should only contain a description "doc" at maximum, it will contain a single package **plugin.<unique package>.** The unique package is usually the plugin **<main package>** but if the main package is the same for multiple plugins we have to use a composition of this main package because the unique package within the plugin needs to be unique among all plugins, also this package name is used as the plugin id. For example for **introspection** we have two plugins one is the standard introspection which provides the components and plugins that are active in the current distribution and the second one is the introspection for the possible requests that can be made, for this two casses the name of the unique package is **plugin.introspection** for the first one and **plugin**_*__**.introspection_request** for the second one event though the main package for both plugins is **introspection**. In this package will be found all modules that contains configuration. Any subpackage will not be considered by the IoC container.
The module **plugin.<unique package>.init** should contains this global variables:

- **NAME**
  The user friendly name for the plugin, this should be a short descriptive string.
- **GROUP**
  The group of the plugin defines a collection of plugins that have a common purpose.
- **VERSION**
  Defines the plugin version. This is a string of the form '1.2' where 1 is the major version of the plugin and 2 is the minor version. The minor version changes whenever the plugin has a release and meanwhile there was some bug fixes performed with no major impact. The major version changes whenever there are significant changes to the plugin and the plugin has already a released version.
- **DESCRIPTION**
  A description of the plugin.

The plugin package "<main package>"

The main package also needs to contain two sub packages that are recognized by the application deployer, This packages can be contained within other packages the considered criteria is the last package name.

- **<main package>.******.api**
  The API package (the ** means that it can be in any package structure) contains all the modules that define the REST API models and services. This package should only contains this modules. Whenever another plugin uses only modules from this package from another plugin is understood that the dependency between the plugins is a light dependency. A light dependency means that if the application should be distributed on several machines this two plugins can run on different boxes using different databases and communicate between them through the API. Basically the plugins can run on different machines with different databases and perform there tasks as if the two plugins are running on the same machine.
- **<main package>.******.meta**
  The meta package contains modules that enhance the REST models. At this point this are the database mappings of the models. Just like in the **api** package case whenever another plugin uses modules from this package a dependency is formed but now is a medium

one. A medium dependency means that the plugins can run on different machines but they have to use the same database since most likely they might use the same tables.

Any other modules that are defined in the plugin and are not in this packages will create a heavy dependency. The heavy dependency means that the plugins need to be part of the same application distribution.

A sample can be found **here**, remember that the egg is simply a zip file.