Hierarchical projects support

Introduction

Expansion of abstraction: full name

Jenkins Item model long allowed a hierarchical structure for projects. This started around 1.312, about 3 years ago. Because the use of hierarchy was limited to small corners (such as matrix projects and Maven projects), it's been often treated as second class citizens.

This initial support for a hierarchical project structure mostly meant differenciating a name of an Item (`getName()`) (or short name) from its full name (`getFullName()`). A full name is just a '/'-separated concatanation of the short names in the path, such as "foo/bar/zot". There are a bunch of methods that deal with full names, such as jenkins.getItemByFullName(name). ItemGroup and Item is the key abstraction for building a hierarchy.

Plugins that do not distinguish full names from short names weren't too big of an issue, because there are not many occasions where you need to refer to a specific module in a Maven project, or a specific configuration in a matrix project.

Expansion of abstraction: contextual name resolution

Then starting around 1.406, which is about a year ago, we made more improvements in this area to make hierarchical projects more useful. Namely, the introduction of *contextual name resolution*. That is, just like "foo.txt" in the file system would resolve to different file depending on your current directory, we'd like the job name "foo" to resolve to a different AbstractProject depending on the current job we are talking about, which we call the context.

So several methods are added to enable this contextual name resolution. They all take ItemGroup as a context — think of it as a current directory. The notion of "item path name" is introduced, which is the equivalent of file path name. An item path name can be resolved to the actual Item by supplying a context ItemGroup.

The syntax of item path name is also modeled after the file system path name. Paths like "foo/bar/zot" or "foo" are relative from the current context, and paths like "/foo/bar/zot" are interpreted as absolute. The only difference is that if a relative "foo/bar/zot" doesn't resolve to anything, the system attempts to resolve "/foo/bar/zot" for backward compatibility.

Call for Action

Today, most plugins do require minimum of 1.424 (the previous LTS base release), but when it comes to proper hierarchical project support, they are built with pre-1.312 assumptions. This prevents more modern plugins, such as cloudbees folder plugin, to take advantages of the expanded abstractions.

This page is created to call an attention to this, and to discuss how you can migrate away from those deprecated methods.

How to modernize the code

To retrieve a list of projects from names, use Items.fromNameList(itemGroup, names, type).

To retrieve a project from a name use jenkins.getItem(relativeName, itemGroup, type). First parameter name may look confusing, but this method supports path names in general, including absolute names like "/foo/bar".

In both cases, the main question is to think what your contextual `ItemGroup` is, and where to get it from. Here are typical ways you do it:

From builds and projects

If you are in the context of a build, the project that owns that build is the right context: `build.getProject().getParent()`

From Descriptor

If you are writing AJAX method, form validation methods, and so on on your Descriptor, you can access the contextual ItemGroup by adding `@AncestorInPath ItemGroup context` to your parameter list.

(`@AncestorInPath` annotation refers to the nearest ancestor object of the given type in the URL that led to the call to your method, so for example you can instead do `@AncestorInPath AbstractProject project`)

From constructors of Describable

Most Describables (such as Builder or JobProperty) are long-running instances. Therefore it is not a good idea to resolve names to Item in your constructor. The job that the name refers to might get renamed, or might not be created/loaded yet.

Instead, keep the name as name, then attempt to resolve it later when you actually need it.

When there's no context

In some places, there really are no contextual ItemGroup. A good example of this is a CLI `build` command. In those cases, you should supply the root `Jenkins` object as the context, to make all relative path names resolve like absolute path names.

Oops, I've used '/' already

Some plugins have already used '/' as a separator between item name and something else. For example, copy-artifacts plugin had used '/' to separate the job name from extra information one can attach to it (see this)

In this case, you should choose a different delimiter (':' is a good delimiter. Any other characters that's not allowed in file system would do, too), then define a separate field to store the value that uses the new delimiter. For example,

```
class Foo {
    /*
     * this is where you used to store "foo/N"
     * @deprecated
     */
    private transient String data;

    // this is where you now store "foo:N"
    private String newData;

    protected Object readResolve() {
        if (data!=null && newData==null)
            newData = data.replace('/','#');
        return this;
    }
}
```

The `readResolve` method gets called when you are resurrected from the persisted state, so you do the data conversion there. In this way, you can change the delimiter without breaking existing configurations. See Hint on retaining backward compatibility for more details.

Examples

Here are some samples of changes done to support hierarchical projects:

- downstream-ext-plugin
- parameterized-trigger-plugin
- join-plugin