Fresh Object and Asset Post-Loading

There is a three-way collision in the way that Fresh currently thinks about post-loading objects.

1. Objects can override Object::loadFromXml(), call the super, then add additional loading code.
2. Objects can override Object::postLoad(), which is called iff the object is being loaded as part of a manifest.
3. Assets can implement an AssetLoader, which can add new properties and control the whole process of loading an asset.

Connected to all of this is the manner in which objects are created. Again, it’s varied.

1. Directly through new <Object> or ObjectManager::createObject().
2. Indirectly through ObjectManager::loadManifest().
3. Semi-directly through Asset::createOrGetAsset().

Roughly, each of the three create methods corresponds to each of the three post-load methods.

The first create method uses loadFromXml() but does not (and *cannot*) call postLoad(). It *cannot* because loadManifest(), which necessarily calls postLoad(), creates many objects, and calling postLoad() for each object defeats the purpose. The purpose of postLoad() is to allow objects to perform post-loading events specifically that rely on the loading and fixup of other objects in the manifest. (Perhaps a better name would be postLoadManifest().) Therefore postLoad() is not helpful for individual object loading.

For that purpose (that is, for post-loading that does not rely on other objects being loaded and fixed up), Object::loadFromXml() should be overridden and extended.

AssetLoader offers the most powerful control over loading, but it only applies to Assets.

That’s the situation, but why is it problematic? There are three problems.

1. I’ve gotten in the habit of putting all post-loading into postLoad(). I should move “individualistic” post-loading into loadFromXml(). Otherwise it will be skipped for individual object creation (like “new <Object>”).
2. There is some ambiguity as to whether a clean division between “individualistic” post-loading and “communal” post-loading is really possible. I think it is, but I’m not sure.
3. Objects often want to add pseudo-properties (“post-properties”). Pseudo-properties prevent good error-detection when an invalid property is added to a object in the XML file. AssetLoaders offer a good solution for this, except that this solution can’t reasonably applied to all Objects. So: how do we support “load-only properties” without causing warnings from loadFromXml()?

This last question is the only one needing real attention here.

There are a number of options.

One option is to use *real* properties where load-only properties are tempted. For example, you could have a vector of elements that is loaded from XML in the usual way, but then deleted immediately after post-loading. This offers the maximum consistency and power. The downside is that these load-only variables do actually live in the object at least to *some* extent throughout its lifetime, and therefore are not truly load-only variables. So there is a mild space efficiency problem.

Another option is to stick with post-load properties as they are now (informal), and simply ignore warnings about them. This would make the warning log noisy and far less useful.

I could also add some sort of actual system to register load-only properties through a formal mechanism, but then automatically clean them up after loading. This sounds difficult (e.g. do you clean them up after loadFromXml() or after postLoad()?).

The first option sounds best. In exchange for a generally small leftover memory cost per object, it gives all the power of the properties system. It only has the downside that load-only properties hang around after loading.

# Update Feb 13, 2012

Postloading continued to be a problem throughout the development of the House of Shadows prototype. It gave me many bugs from the beginning right up to the end. It needs to be fixed.

The essential problem is complexity. postLoad() itself works great, so far as it goes. It has become clear that another layer of postloading is also necessary.

First, note that because objects are created outside of manifest loads, and because postloading is instrumental in object creating, even objects that are created “loose” (not as part of a manifest load) should be postloaded.

Manifest loading is very useful. Therefore, objects sometimes want to load manifests as part of their own creation. But this doesn’t currently work, because the postload process assumes that only one postloading process can happen at a time. Nested manifest loads are not allowed. It would be helpful if I relaxed this restriction.

The purpose of postloading is to give objects a chance to complete their construction after pointer fixup. But it has become clear that another layer of post-loading is sometimes helpful. For example, an object may want to deal with another object only after the other object has postloaded. Since order of postloads isn’t guaranteed, there is no reliable way to do this. Consequently I’ve added an HOS-specific “beginLevel()” call that is called for every Actor as the level is beginning. This solution is adequate, but it sprang up in an ad-hoc way and is therefore confusing and muddled.

The most miserable test case for this mush of initialization events is the Player—especially its headlight and flashlight. For some reason the headlight is created in onAddedToParent() (a kind of ad-hoc post-post-load), but the flashlight is added in beginLevel(). This is brittle and troublesome. Concretely, it prevented me easily attaching the headlight to the flashlight a few days ago. And it’s an error-prone mess. I often get errors where an object has an access failure on an other object because the other object hasn’t been constructed yet or whatnot.

So I need a better solution.

The main challenge in devising a solution is dealing with the problem of infinite regression. Clearly I could add a third event:

loadFromXml()

postLoad()

postPostLoad()

(or more seriously, loadFromXml(), postFixup(), and postLoad())

But how long would it be until I needed another layer to do initialization that requires post-post-load to have executed? Indeed, the onAddedToParent() approach (which has the advantage of executing even for non-disk-loaded objects, or objects that travel between levels) represents a fourth layer that is currently in use.

I could do a single post-post-load (or post-load?) where the objects specify their own sort position. That would at least prevent the infinite regression problem. In the theoretical case where two objects need to go back and forth in their post-initialization, I could handle that in a bespoke manner.

Another complication that I forgot to mention is just the mechanical problem of allowing objects to create other objects during their postLoad. This is a case of the container-iterator-switcheroo problem. For lists it results in objects being double-initialized or skipped. For vectors it results in crashes. I need to handle it gracefully in a single place, with something like thoroughTraverse() (see Namespace::recurse()). This isn’t a problem with postLoading per se—rather a problem with any algorithm that traverses a container while the container might be modified.