XML in SmootLight: Configuration++

Russell Cohen

February 16, 2011

1 Motivation

Why use XML (or any non-code language for that matter) to configure code? 2 Reasons:

- We would like small changes (like changing the color, speed, or type of behavior) to be as quick as possible, and require modifying only 1 piece of code.
- We would like these changes to be able to be made *programmatically*
- (Not applicable to python, but important in languages like Java or C): We want to be able to make changes **without** having to recompile the source.

As you will see, however, XML in SmootLight goes beyond simple configuration. XML in SmootLight allows us to declare a LightSystem (an inherently non-declarative thing) in the same way you might write a webpage in HTML. We will refer to the XML system here-on-in as 'SmootConf'. Without any further ado, lets start looking at how this all works.

2 Declaring a class in SmootConf

The most common thing done is SmootConf is declaring a class – Class declaration code will get parsed by SmootLight at runtime and *actually* **declare** your classes for you, exactly how you describe them. Classes are declared under a broader `Configuration` tag which we will describe later. Lets look at the declaration of `PygameInput`, an input that takes data from a Pygame window.

```
<InputElement>
    <Class>inputs.PygameInput</Class>
    <Args>
        <Id>pygameclick</Id>
        <RefreshInterval>10</RefreshInterval>
        <Clicks>True</Clicks>
    </Args>
</InputElement>
```

The first attribute we see is the `Class` attribute. This specifies what fully-qualified Python class to this object should be an instance of. In this case, it is an instance of PygameInput, which lives in the inputs module/folder. Next, we see the `Args`. The Args are where *every* piece of configuration (except the actual Class) goes. Let me repeat that, becuase it is a common sticking point. If you place something in the configuration outside of the `Args` tag, it will not be read. Period.

If you are familiar with the SmootLight system, you will know that many objects in SmootLight behave like dictionaries – you can use statements like `Self['ParamName']` to access parameters. If you have ever wondered where this mystery dictionary is filled from, look no further – it is here in the Args tag.

Lets dig into the contents of the Arg tag. First we see `Id`. All components in the SmootLight system are *not* explicitly required to have an Id specified.[1] However, if you want to be able to reference this class in other places in the XML (which we will look into later), you will need to specify and Id. The other two parameters are simply bits of configuration which will get passed to PygameInput when it gets instantiated.

3 The Structure of a SmootLight Configuration Document

The individual class declarations are the 'leaves' of a full configuration document that gets interpreted by the parser. In order for the parser to know what to do with them, it also needs the branches. The structure of these 'branches' (tags) follow:

```
<LightInstallation>
    <InstallationConfiguration>
        <Defaults />
    </InstallationConfiguration>
    <PixelConfiguration />
    <PixelMapperConfiguration />
    <RendererConfiguration />
    <InputConfiguration />
    <BehaviorConfiguration />
</LightInstallation>
```

Under each of the tags indicated, place the classes that you want to instantiate in that category. Each category has a different child tag:

- PixelConfiguration: PixelStrip
- PixelMapperConfiguration: PixelMapper
- RendererConfiguration: Renderer
- InputConfiguration: InputElement
- BehaviorConfiguration: Behavior

Some further clarification on this: Recall in the previous section we inspected the declaration of the Input element. The input configuration for that system might have looked like:

```
<InputConfiguration>
    <InputElement>
        <Class>inputs.PygameInput</Class>
        <Args>
            <Id>pygameclick</Id>
            <RefreshInterval>10</RefreshInterval>
            <Clicks>True</Clicks>
        </Args>
    </InputElement>
</InputConfiguration>
```

[1]Components declared without Id's will get a randomly assigned Id at declaration time