1 SmartFrog Component Autoloader

The autoloader provides the facility to load components on demand, when other components attempt to access them through a reference. The autoloader intercepts reference resolutions and will deploy the resource associated with a specific reference if it has not already been deployed, and then complete the reference resolution.

1.1 Architecture

The Autoloader is a component that is a modified `Compound`. It intercepts references that resolve through it and examines the references to see if the reference requires a description to be downloaded and deployed. First it attempts to resolve the reference, If it fails, it examines the next reference part to be resolved which must be a "`HERE`" reference part indicating an attribute name within the autoloader's context – any other reference part is considered an error and the original exception is returned. If the attribute does not yet exist, it will validate the attribute name that needs to be present against both a validation pattern and an optional list of valid names and if OK, will attempt to deploy a resource identified by the name combined with a number of other strings: specifically a prefix, a postfix and a language extension. Once the resource has been successfully loaded, the resolution of the reference is repeated and the result returned (including any further resolution exception).

The component cannot be used to trigger a remote deployment as it will cause a lock-up with the synchronization – it should only be used to deploy components into the JVM in which the autoloader resides.

More than one autoloader my be used within a single JVM.

1.2 Implementation

The implementation is as a single component "`org.smartfrog.services.autoloader.Autoloader`". There is a single SmartFrog component template, contained in the file `"/org/smartfrog/services/autoloader/components.sf"` which provides the basic Autoloader template for instantiating an autoloader.

The autoloader class is a subclass of `Compound` in which the `sfResolve(reference, index)` method has been over-ridden to provide the required autoloading capabilities.

1.3 Component

1.3.1 Autoloader

This is a component whose behaviour is controlled by the following attributes:

Attribute Name	*Description*	*Optional/Mandatory*
`matches`	A string defining rules for the matching of names before they will be accepted as valid. The string is a Java Pattern (`java.util.Pattern`)	Optional, default `"\\w+"`, allowing names containing 1 or more of the characters[a-`zA-Z0-9_`]
`validLoads`	A vector which if set, defines a list of valid names to be requested. If not set, the test is not applied.	Optional: default null

Attribute Name	*Description*	*Optional/Mandatory*
URLPrefix	A prefix string for building the resource to be loaded by autoloader. The prefix is often used to define a directory in which to find the resources, The resource name is generated from the attribute name as follows: `URLPrefix + attrName + URLPostfix + "." + language`	Optional: default ""
URLPostfix	A postfix string for building the resource to be loaded by autoloader.The resource name is generated from the attribute name as follows: `URLPrefix + attrName + URLPostfix + "." + language`	Optional: default ""
language	A language extension string for building the resource to be loaded by autoloader. The resource name is generated from the attribute name as follows: `URLPrefix + attrName + URLPostfix + "." + language`	Optional: default "sf"

1.4 Example

A typical usage will be to add the autoloader into the `default.sf` file so as to prepare the JVM with autoloaded services that can be accessed if required by further deployments.

```
default.sf:
#include "/org/smartfrog/services/autoloader/components.sf"

sfConfig extends Compound {
    ... // whatever is there normally
    autoloader extends Autoloader {
        sfProcessComponentName "autoloader";
        URLPrefix "/a/directory/path/";
    }
}
```

And in a file to be deployed using `sfStart`:

```
sfConfig extends ... {
    ...
    foobarService LAZY HOST localhost:autoloader:foobar;
    ...
}
```

When the reference `foobarService` is dereferenced, the file `"/a/directory/path/foobar.sf"` will be deployed and the appropriate value returned.