File Components

The file components provide a cross platform way of representing files. There is an underlying pattern to most of these components, in that after the component is running, it sets the attribute `absolutePath` to the platform-specific, absolute path of the file or directory in question. Other components can bind to this value, merely by setting a `LAZY` link to the `absolutePath` attribute of a component.

This same technique is used in other components, such as the repository download components of the Java support package; again, at runtime, the `absolutePath` attribute points to the relevant file -in this example the local cached copy of the downloaded JAR file. Any component that provides filenames to other components should strongly consider adopting this pattern.

`File`	Describes a file, with optional liveness checks for existence and type
`Mkdir`	Creates a directory
`SelfDeletingFile`	Identifies a file that must be deleted when the application is terminated
`TempFile`	instantiates a temporary file
`TempDir`	Creates a temporary directory
`TextFile`	Saves text to a named file, with optional encoding
`TouchFile`	Sets the timestamp of a file, creating it if needed.
`CopyFile`	Copies a file

Declaring the components

The components are implemented in the package `org.smartfrog.services.os.filesystem`. To use them in a deployment descriptor,

```
#include "/org/smartfrog/services/filesystem/components.sf"
```

This will include the schema and component descriptions ready for use.

Attributes Common to most components

name	***type***	***description***
`filename`	String or Component	Name of a file, or a reference to component that implements `FileIntf`, in which case the method `FileIntf.getAbsolutePath()` will be used to get the absolute path of the file.
`deleteOnExit`	boolean	Attribute for those components (`SelfDeletingFile`, `TempFile`, `TextFile`), that can be deleted on termination.
`absolutePath`	read only string	the absolute, platform specific path of the file. Equivalent to `java.io.File.getAbsolutePath()`
`URI`	read only string	A file: URI to the file. Equivalent to `java.io.File.getURI()`

Detach and Termination Attributes. Common to File, Mkdir and TouchFile components

Components that support this attributes: `File`, `Mkdir`, `TouchFile`.

name	type	description
`sfShouldDetach`	`OptionalBoolean`	Should the component detach itself from its parent after completing its sfStart livecycle method.
`sfShouldTerminate`	`OptionalBoolean`	Should the component terminate itself from its parent after completing its sfStart livecycle method.
`sfShouldTerminateQuietly`	`OptionalBoolean`	Should the component terminate quietly (it does not notify its parent) when it terminates.

All these attributes are optional and can be combined. Example: a component can detach itself and then terminate. These attributes are particularly useful when the components are used in workflows.

Components

File

This component represents a file. It does not have any actions at during deployment or termination, other than to:

1. Convert the parameters describing the file into a platform specific format.
2. Set the `absolutePath` and `uri` attributes, as with other filesystem components
3. Set the other read-only attributes to the state of the file/directory.

It can respond to liveness checks by verifying that any declarations about the state of the file still hold.

There are three ways of using this component. First, it can be used to identify files to work with. Secondly, it can take existing files, and apply liveness checks to the file. Thirdly, by converting OS operations that query the file into component attributes, it can be used to feed file state information into other components.

Writeable attributes

name	type	description
`filename`	`String`	Name of a file
`dir`	`String or Component`	Directory
`mustExist`	`OptionalBoolean`	file must exist
`mustRead`	`OptionalBoolean`	the process must have read access
`mustWrite`	`OptionalBoolean`	the file must be writeable
`mustBeFile`	`OptionalBoolean`	must be a file
`mustBeDir`	`OptionalBoolean`	must be a directory
`testOnStartup`	`OptionalBoolean`	verify state of file during startup
`testOnLiveness`	`OptionalBoolean`	verify state of file in liveness checks

It also supports `OptionalBoolean` attributes: `sfShouldDetach`, `sfShouldTerminate and sfShouldTerminateQuietly.`See the "Detach and Terminate Attributes" section for more information.

Read-only attributes

name	type	description
`absolutePath`	read only string	the absolute, platform specific path of the file. Equivalent to `java.io.File.getAbsolutePath()`
`uri`	read only string	A file: URI to the file. Equivalent to `java.io.File.getURI()`

name	***type***	***description***
`exists`	`Boolean`	true iff the file exists
`isFile`	`Boolean`	true if the file is
`isDirectory`	`Boolean`	true if the file is a directory
`isHidden`	`Boolean`	true if the file is
`timestamp`	`long`	timestamp of the file, -1 if the file is not present
`length`	`long`	length of file (0 if the file is not present)
`isEmpty`	`Boolean`	true if the file is of length zero (or implicitly: does not exist)

SelfDeletingFile

This component deletes a file when it is terminated. If the file does not exist, or the `deleteOnExit` flag is not set to `true`, this does not take place.

name	***type***	***description***
`filename`	String or Component	Name of a file, or a reference to component that implements `FileIntf`, in which case the method `FileIntf.getAbsolutePath()` will be used to get the absolute path of the file.
`deleteOnExit`	`Boolean`	Attribute for those components (`SelfDeletingFile`, `TempFile`, `TextFile`), that can be deleted on termination.
`absolutePath`	read only string	the absolute, platform specific path of the file. Equivalent to `java.io.File.getAbsolutePath()`
`uri`	read only string	A file: URI to the file. Equivalent to `java.io.File.getURI()`

TempFile

This component names a temporary file.

name	***type***	***description***
`prefix`	`String`	Prefix -this should be three or more characters long
`suffix`	`OptionalString`	suffix, e.g. ".tmp"
`dir`	`OptionalString`	a directory. If not specified, the temp directory for this JVM will be used.
`deleteOnExit`	`Boolean`	Attribute for those components (`SelfDeletingFile`, `TempFile`, `TextFile`), that can be deleted on termination. Default=false.
`absolutePath`	read only string	the absolute, platform specific path of the file. Equivalent to `java.io.File.getAbsolutePath()`
`uri`	read only string	A file: URI to the file. Equivalent to `java.io.File.getURI()`

Files created with `deleteOnExit` set to true, it will be deleted when the component terminates. If the file cannot be deleted at that point in time, the file is marked `deleteOnExit` for the JVM itself to clean up if it shuts down cleanly. This is an emergency measure which cannot be relied upon.

TempDir

This component is a variant of `TempFile` (with exactly the same attributes), which instead creates a temporary directory. The directory is created and the `absolutePath` and `uri` attributes set to point to it.

If the directory is created with `deleteOnExit` set to true, it and all child directories and files will be deleted when the component terminates. If the directory already existed when deployment took place, it is not deleted during termination. This is a safety feature designed to stop users accidentally deleting their home directory or some other valued piece of the filesystem.

TextFile

A text file

name	*type*	*description*
`filename`	`String` or `Component`	Name of a file, or a reference to component that implements `FileIntf`, in which case the method `FileIntf.getAbsolutePath()` will be used to get the absolute path of the file.
`deleteOnExit`	`Boolean`	Request deletion on termination.
`absolutePath`	read only string	the absolute, platform specific path of the file. Equivalent to `java.io.File.getAbsolutePath()`
`uri`	read only string	A file: URI to the file. Equivalent to `java.io.File.getURI()`
`encoding`	`String`	Text encoding to use (default="`utf8`")
`text`	`String`	Text to write

When a `TextFile` component is deployed, it fills in the nominated file with the contents of the `text` attribute, using whatever encoding is requested. The file will be deleted at termination, if `deleteOnExit` is set.

TouchFile

This component touches a file. if the file does not exist, it is created. A timestamp can be passed in as seconds since 1970-01-01, or -1 for "latest time".

```
sfConfig extends Compound {
    sfSyncTerminate true;

    temp1 extends TempFileWithCleanup {
        deleteOnExit true;
        prefix "temp1";
        suffix ".txt";
    }

    assert extends Assert {
        fileExists LAZY temp1:filename;
    }

    touch extends TouchFile {
        filename PARENT:filename;
        timestamp PARENT:timestamp;
    }

    //the filename
    filename LAZY temp1:absolutePath;
    //and timestamp
    timestamp 100000L;
}
```

It also supports `OptionalBoolean` attributes: `sfShouldDetach`, `sfShouldTerminate and sfShouldTerminateQuietly.`See the Detach and Terminate Attributes section.

CopyFile

This component creates a copy of a file.

name	type	description
`source`	`FilenameType`	Either a string filename or a File component (or other component that has or sets the attribute `absolutePath`).
`destination`	`FilenameType`	Either a string filename or a File component (or other component that has or sets the attribute `absolutePath`).

```
#include "/org/smartfrog/services/filesystem/components.sf"

sfConfig extends CopyFile {
 source extends File {
    //a directory
    dir "/";
    //file must always exist
    mustExist true;
    testOnDeploy true;
    filename "test.sf";
 }

 // The copy will be a SelfDeletingFile.
 // The copy will be deleted when CopyFile terminates
 destination extends SelfDeletingFile {
    filename "testSelfDeleteCopy.sf";
 }
}
```

It also supports OptionalBoolean attributes: `sfShouldDetach`, `sfShouldTerminate and sfShouldTerminateQuietly.`See Detach and Terminate Attributes section.

DeployOnCopy

This is an extension of the CopyFile component that deletes the copied file when terminating. It can be used to deploy to any application server that automatically deploys any file copied into its deployment directory. The supported attributes are those of CopyFile,

```
#include "/org/smartfrog/services/filesystem/components.sf"

sfConfig extends DeployOnCopy {
 source  "/home/example/app/dist/lib/application-3.14.war";


 destination "/home/example/jboss/server/default/deploy/application.war";

}
```

If the copy failed, then the destination file is *not* deleted during termination.

Mkdir

This component creates a directory when deployed. All necessary parent directories are auto-created.

name	type	description
`dir`	`String` or `Component`	Name of a file, or a reference to component that implements `FileIntf`, in which case the method `FileIntf.getAbsolutePath()` will be used to get the absolute path of the file.
`parent`	`OptionalString` or `Component`	Parent directory. Optional
`deleteOnExit`	`Boolean`	`Request deletion on termination`
`absolutePath`	read only string	the absolute, platform specific path of the directory. Equivalent to `java.io.File.getAbsolutePath()`

name	*type*	*description*
`uri`	read only string	A file: URI to the directory. Equivalent to `java.io.File.getURI()`

If the directory is created with `deleteOnExit` set to true, it and all child directories and files will be deleted when the component terminates. As a safety check, if the directory existed before deployment, it will NOT be deleted. This is to reduce the risk of accidentally deploying something that deletes a user's home directory, or similar.

It also supports `OptionalBoolean` attributes: `sfShouldDetach`, `sfShouldTerminate and sfShouldTerminateQuietly.`See the Detach and Terminate Attributes section.

Example: Mkdir

```
#include "/org/smartfrog/services/filesystem/components.sf"
#include "/org/smartfrog/services/assertions/components.sf"

sfConfig extends Compound {

    newdir LAZY mkdir:absolutePath;

    sfSyncTerminate true;

    mkdir extends Mkdir {
        parent LAZY PROPERTY java.io.tmpdir;
        dir "/new-directory-for-mkdir";
        deleteOnExit true;
    }

    assert extends Assert {
        dirExists PARENT:newdir;
    }

}
```

This example creates a directory under the parent directory `${java.io.tmpdir}`, then asserts that it has been created. Note the use of LAZY PROPERTY reference when extracting this value. If the non-lazy property were used, the parent attribute would be set to the temporary directory of the JVM/Process which parsed the deployment descriptor, *not* the process which actually deployed the component. When deploying to a remote system, the difference can be significant.

Although "/" is used as the directory separator, this descriptor is still valid on Windows systems, and other platforms with alternate path separators. The directory attribute will have / and \ characters converted to the local platform's type during deployment. The target platform is not an issue with the file types, although the value of the `absolutePath` attribute will be different for the different systems.

Limitations of the components

1. Because Java has no explicit access to file system permissions, SmartFrog components cannot create files with access rights other than the default for the Java process. Java1.6 promises to correct this; until then you need to execute the native `chmod` program.
2. There is not (yet) an rmdir component, to delete a directory.

Examples

Example: temporary text file

This is a temporary text file that is deleted after termination

```
#include "/org/smartfrog/services/filesystem/components.sf"
#include "/org/smartfrog/services/assertions/components.sf"

sfConfig extends Compound {
    sfSyncTerminate true;

    temp1 extends TempFile {
        deleteOnExit true;
        prefix "temp1";
        suffix ".txt";
    }

    assert extends Assert {
        fileExists LAZY temp1:absolutePath;
    }

    textFile extends TextFile {
        file LAZY temp1;
        text "Here is some text that we want to use in our document";
    }

    //the filename
    absolutePath LAZY textFile:absolutePath;
    //the uri
    uri LAZY textFile:uri;
}
```

The `temp1` component names and creates a temporary file in the system's temporary directory. The text file component then fills this in with some text of our choice, in the default (UTF8) encoding.

The a`ssert` component verifies that the file exists;

The `absolutePath` attribute in the root component is LAZY bound to the value of the `textFile`. This component is not explicitly set, but is implicitly set when the component binds to the `file` component. This happens at deployment time. The `uri` attribute is similar.

Because the `temp1` file is already marked as `deleteOnExit`, there is no need to indicate this in the `textFile` declaration, though to do so should be harmless. We say should, as the sole risk is that during undeployment, after `temp1` deletes the file a new file may be created with the same name as is about to be deleted, a file that `textFile` may then unwittingly delete. This is a possible, albeit unlikely race condition.

Example2: encoded text file

This example uses a different text encoding, and an alternate cleanup mechanism

```
#include "/org/smartfrog/services/filesystem/components.sf"

sfConfig extends Compound {

    sfSyncTerminate true;

    temp1 extends TempFile {
        prefix "encoded";
        suffix ".txt";
    }

    cleanup extends SelfDeletingFile {
        file LAZY temp1;
    }

    textFile extends TextFile {
        file LAZY temp1;
        text "UTF16";
        encoding "UTF-16";
    }

    //the filename
    absolutePath LAZY textFile:absolutePath;
    //the uri
    uri LAZY textFile:uri;
}
```

Here, a `SelfDeletingFile` is used to clean up the file at termination time.

Using the filesystem components in other components

The goal of these tasks is to make it easy to name files in a cross platform manner.

Here are the ways to do this.

Extend FileUsingComponentImpl

This class has support code for the core writeable attributes (`file`, `deleteOnExit`), and those that are set at runtime (`absolutePath`, `uri`). To use the features

1. extend the class `FileUsingComponentImpl`.
2. In `sfDeploy()` or later, bind to a filename.
3. If `deleteOnExit` is to be supported, call `deleteFileIfNeeded()` during termination.
4. Implement any other interfaces or operations that are desired. Note that the methods of `FileIntf` and `UriIntf` are already implemented.

To bind to a filename

- use `bind(File)` to set the runtime attributes, and set the `file` member variable, a variable that can be accessed via `getFile()`;
- Use `bind(boolean mandatory,String defval)` to force the `filename` attribute to be read, converted from a File instance or a string path into an absolute path, and then bound to.
- Determine the file name as a string, and use `setAbsolutePath(String)` to bind the component to a path.

Use static helper methods in FileSystem

There are is a static method, `lookupAbsolutePath()`, in the class `FileSystem`, methods that can resolve any attribute of a named component, and then either convert its string value into a local pathname, or resolving it to a `FileIntf` interface, ask for the path with a call to `getAbsolutePath()`. The `resolveAbsolutePath()` method does the same, except it returns a `File` instance.

The `FileSystem` class also includes helper methods to close input and output streams quietly, without throwing an IO exception, and checking for null parameters. These should be used in exception handlers, to quietly close streams on failure. They should not be used in the main body of a method, as there may be a valid reason for a close operation to fail (such as a full filesystem), valid reasons that should be propagated.

Consult the Javadoc documentation for details on how to use these method. It can be used from any component that needs to resolve pathnames.