File Components

The file components provide a cross platform way of working with files.

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
Chmod	Change file permissions on Unix systems
FileExists	A condition to use in conditional components, which tests for a file existing, and allows extra criteria to be asserted.
Files	A component that builds a list of files in a directory, based on pattern matching.
FilesCompound	A compound component that aggregates multiple files from its children, to create a duplicate-free file list.

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, Chmod 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 lifecycle method.
sfShouldTerminate	OptionalBoolean	Should the component terminate itself from its parent after completing its sfStart lifecycle 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/creates a temporary file.

name	type	description
prefix	String	Filename prefix -this should be three or more characters long
suffix	OptionalString	Filename suffix, e.g. ".tmp"
createOnDeploy	Boolean	Should the temp file be created at deploy time (for early file attributes), or when the component is officially started (good when the contents or paths come from other components)
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()`

name	type	description
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.

TextFile

A text file is a text file that is created on deployment, and optionally, deleted afterwards.

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.

name	type	description
parent	Optional String 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()
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 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 a 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.

Chmod

This component executes the Unix command chmod to set the permissions of an existing file.

name	type	description
file	String or Component	Name of a file, or a LAZY reference to file component
recursively	Boolean	Should the operation be recursive? Default: false
octalCode	String	An octal code such as "777" or "700"
userPermissions	String	User permissions such as "+r+w-x"
groupPermissions	String	Group permissions such as "+r+w-x"

name	type	description
othersPermissions	String	Group permissions such as "+r+w-x"

The component supports the workflow attributes: sfShouldDetach, sfShouldTerminate and sfShouldTerminateQuietly. See Detach and Terminate Attributes section.

Example: Chmod

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

      rmAllRights extends Chmod {
          file "/tmp/some-file.txt" ;
          octalCode "000";
          sfShouldTerminate true;
        }
```

This example declares that the file "/tmp/some-file.txt" is to have all permissions removed from it; not even its owner will be able to read this file.;

Example: Chmod

```
testfile extends TempFileWithCleanup {

}

setWriteable extends Chmod {
    sfShouldTerminate true;
    file LAZY testfile;
    userPermissions  "+w";
    groupPermissions "+w";
    otherPermissions "+w";
}
```

This example declares a temporary file, and then makes it writeable by everyone.

FileExists

The FileExists component is a condition which can be used to test/assert that a file or directory exists.

name	type	description
file	String or Component	Name of a file, or a LAZY reference to file component
minSize	Boolean	Should the operation be recursive? Default: false
canBeDirectory	String	Indicates the file can be a directory. Default: true
canBeFile	Boolean	Indicates the file can be a simple file. Default: true

The extension class, DirExists simply declares that the file cannot be a simple file.

```
DirectoryExists extends FileExists {
  canBeFile false;
}
```

The condition can be used in any of the conditional workflow components, including the test compounds. The state of the file is assessed every time the condition is evaluated.

Files

This is a component that builds a list of files in a directory, based on pattern matching. Some experience with the java.utils.regex package is strongly encouraged before attempting to use this component.

It is defined in the file /org/smartfrog/services/filesystem/files/components.sf :

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

name	type	description
dir	String or Component	Name of a directory to scan in, or a LAZY reference to a file component identifying a directory
pattern	String	Regular expression. Default: "\\w+"
includeHiddenFiles	Boolean	Should hidden files be included. Default: false
caseSensitive	Boolean	Should the pattern be case sensitive?
fileCount	Integer	Exact number of expected files; -1 (the default) means "don't check"
minFileCount	Integer	Minimum number of matching files; -1 (the default) means "don't check"
minFileCount	Integer	Maximum number of expected files; -1 (the default) means "don't check"

The fileCount, minFileCount, and minFileCount attributes exist to make component testing easier, but also to provide sanity checks on operations. By setting a high limit (such as 999) to the minFileCount attribute, one can prevent operations acting on too many files, and so detect when the operation is being applied to the wrong directory. Similarly by setting the minFileCount attribute to a positive integer, one can declare a minimum number of files that the pattern should match against.

After being started, the component updates three attributes with values based on the files matching the pattern:

name	type	description
fileSet	Serialized fileset instance	A serialized fileset class, which can be used by other components
fileSetString	String	The set of files converted into a string list, with each entry separated by the platform's path separator.
fileList	Vector	The set of files as a vector of strings.

Note that the fileset is evaluated once, during deployment.

FilesCompound

This is a compound component that aggregates the file attributes of all compatible children, namely

- Any immediate child that implements the Files interface; all its files are picked up.
- Any immediate child that implements the FileIntf interface; its single file is picked up.
- Any child component that has an attribute absolutePath; the value of this attribute is taken as the name of a file.

The component implements the Files interface itself, so FilesCompound instances can be deployed inside other FilesCompound instances. The component also supports the fileCount, minFileCount, and minFileCount attributes to validate the aggregate size of the files. If you use these, remember that duplicate entries are eliminated when the aggregate fileset is built up; the fileCount value may be smaller than expected.

After startup, the three result attributes of Files are also set: fileSet, fileSetString and fileList.

Limitations of the components

- There is not (yet) an rmdir component, to delete a directory.
- We can't make assertions about the permissions of a file, because Java hides it from us

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.

Example 4: locating text files

This example uses the `Files` component to locate text files in the temp directory:

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

textfiles extends Files {
  dir LAZY PROPERTY java.io.tmpdir;
  pattern "\\w+\\.txt";
  minFileCount 2;
}
```

A minimum file count is declared -2- so the component will fail to deploy if less than this number is found in the directory.

Example 3: Aggregating files

This is an interesting example. It uses the `FilesCompound` to aggregate all child files. What is interesting about it is that the `Mkdir` and `TouchFile` components are both components that can be deployed in it -and when this is done, the generated filenames are added to the list. The `fileCount` attribute can then be used to declare the number of files and directories created.

```
BuildTestDirs extends FilesCompound {

    fileCount 8;

    mkdir1 extends Mkdir {
        parentDir LAZY PROPERTY java.io.tmpdir ;
        dir "filestests";
    }
    subdir extends Mkdir {
        parentDir LAZY mkdir1;
        dir "subdir1";
    }
    subdir2 extends Mkdir {
        parentDir LAZY mkdir1;
        dir "subdir2";
    }
    f1 extends TouchFile {
      dir LAZY mkdir1;
      filename "1.txt";
    }
    f2 extends TouchFile {
      dir LAZY mkdir1;
      filename "2.txt";
    }
    f3 extends TouchFile {
      dir LAZY  subdir;
      filename "1.txt";
    }
    f4 extends TouchFile {
      dir LAZY  subdir;
      filename "2.txt";
    }
    f5 extends TouchFile {
      dir LAZY  subdir2;
      filename "1.txt";
    }
}
```

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.