# APS Filesystem Service Provider

User Guide

Version: 1.0.0

Author: Tommy Svensson

Table of Contents

# 1 APSFilesystemService

This provides a filesystem for writing and reading files. This filesystem resides outside of the OSGi server and is for longterm storage, which differs from BundleContext.getDataFile() which resides within bundle deployment. The APSFilesystemService also does not return a File object! It priovides a file area for each unique owner name that is accessed through an API that cannot navigate nor access any files outside of this area. The ”owner” name should be either an application name or a bundle name if it is only used by one bundle.

The APSConfigService uses the APSFilesystemService to store its configurations.

## 1.1 Setup

The *aps.filesystem.root* system property must be set to point to a root where this service provides its file areas. This is either passed to the JVM at server startup or configured withing the server. Glassfish allows you to configure properties within its admin gui. Virgo does not. If this is not provided the service will use BundleContext.getDataFile(".") as the root, which will work for testing and playing around, but should not be used for more serious purposes since this is not a path with a long term availability.

## 1.2 The service

The service allows you to create or get an APSFilesystem object. From that object you can create/read/delete directories (represented by APSDirectory) and files (represented by APSFile). You can get readers, writers, input streams and output streams from files. All paths are relative to the file area represented by the APSFilesystem object.

The javadoc for the APSFilesystemService.

## 1.3 The APIs for this service

public *interface* **APSDirectory** extends APSFile [se.natusoft.osgi.aps.api.core.filesystem.model] {

This represents a directory in an *APSFilesystem*.

Use this to create or get directories and files and list contents of directories.

Personal comment: I do prefer the term "folder" over "directory" since I think that is less ambiguous, but since Java uses the term "directory" I decided to stick with that name.

**APSDirectory createDir(String name) throws IOException**

Returns a newly created directory with the specified name.

*Parameters*

*name* *- The name of the directory to create.*

*Throws*

*IOException* *- on any failure.*

**APSDirectory createDir(String name, String duplicateMessage) throws IOException**

Returns a newly created directory with the specified name.

*Parameters*

*name* - *The name of the directory to create.*

*duplicateMessage* - *The exception messaging if directory already exists.*

*Throws*

*IOException* - *on any failure.*

**APSFile createFile(String name) throws IOException**

Creates a new file in the directory represented by the current *APSDirectory*.

*Parameters*

*name* - *The name of the file to create.*

*Throws*

*IOException* - *on failure.*

**APSDirectory getDir(String dirname) throws FileNotFoundException**

Returns the specified directory.

*Parameters*

*dirname* - *The name of the directory to enter.*

*Throws*

*FileNotFoundException*

**APSFile getFile(String name)**

Returns the named file in this directory.

*Parameters*

*name* - *The name of the file to get.*

**void recursiveDelete() throws IOException**

Performs a recursive delete of the directory represented by this *APSDirectory* and all subdirectories and files.

*Throws*

*IOException* - *on any failure.*

**String[] list()**

*See*

*java.io.File.list()*

**APSFile[] listFiles()**

*See*

*java.io.File.listFiles()*

}

---

public *interface* **APSFile** [se.natusoft.osgi.aps.api.core.filesystem.model] {

This represents a file in an *APSFilesystemService* provided filesystem. It provides most of the API of *java.io.File* but is not a *java.io.File*! It never discloses the full path in the host filesystem, only paths relative to its *APSFilesystem* root.

Use the createInputStream/OutputStream/Reader/Writer to read and write the file.

**InputStream createInputStream() throws IOException**

Creates a new *InputStream* to this file.

*Throws*

*IOException*

**OutputStream createOutputStream() throws IOException**

Creates a new *OutputStream* to this file.

*Throws*

*IOException*

**Reader createReader() throws IOException**

Creates a new *Reader* to this file.

*Throws*

*IOException*

**Writer createWriter() throws IOException**

Creates a new *Writer* to this file.

*Throws*

*IOException*

**Properties loadProperties() throws IOException**

If this file denotes a properties file it is loaded and returned.

*Throws*

*IOException* *- on failure or if it is not a properties file.*

**void saveProperties(Properties properties) throws IOException**

If this file denotes a properties file it is written with the specified properties.

*Parameters*

*properties* *- The properties to save.*

*Throws*

*IOException* *- on failure or if it is not a properties file.*

**APSDirectory toDirectory()**

If this *APSFile* represents a directory an *APSDirectory* instance will be returned. Otherwise *null* will be returned.

**APSFile getAbsoluteFile()**

*See*

*java.io.File.getAbsoluteFile()*

**String getAbsolutePath()**

Returns the absolute path relative to filesystem root.

**APSFile getCanonicalFile() throws IOException**

*See*

*java.io.File.getCanonicalFile()*

**String getCanonicalPath() throws IOException**

*See*

*java.io.File.getCanonicalPath()*

**String getParent()**

*See*

*java.io.File.getParent()*

**APSDirectory getParentFile()**

*See*

*java.io.File.getParentFile()*

**String getPath()**

*See*

*java.io.File.getPath()*

**boolean renameTo(APSFile dest)**

*See*

*java.io.File.renameTo(File)*

**String getName()**

*See*

*java.io.File.getName()*

**boolean canRead()**

*See*

*java.io.File.canRead()*

**boolean canWrite()**

*See*

*java.io.File.canWrite()*

**boolean exists()**

*See*

*java.io.File.exists()*

**boolean exists(String name)**

Checks if the named file/directory exists.

*Returns*

*true or false.*

*Parameters*

*name - The name to check.*

**boolean isDirectory()**

*See*

*java.io.File.isDirectory()*

**boolean isFile()**

*See*

*java.io.File.isFile()*

**boolean isHidden()**

*See*

*java.io.File.isHidden()*

**long lastModified()**

*See*

*java.io.File.lastModified()*

**long length()**

*See*

*java.io.File.length()*

**boolean createNewFile() throws IOException**

*See*

*java.io.File.createNewFile()*

**boolean delete()**

*See*

*java.io.File.delete()*

**void deleteOnExit()**

*See*

*java.io.File.deleteOnExit()*

**String toString()**

Returns a string representation of this *APSFile*.

**File toFile()**

This API tries to hide the real path and don't allow access outside of its root, but sometimes you just need the real path to pass on to other code requiring it. This provides that. Use it only when needed!

*Returns*

*A File object representing the real/full path to this file.*

}

---

public *interface* **APSFilesystem** [se.natusoft.osgi.aps.api.core.filesystem.model] {

This represents an *APSFilesystemService* filesystem.

**APSDirectory getDirectory(String path) throws IOException**

Returns a folder at the specified path.

*Parameters*

*path* *- The path of the folder to get.*

*Throws*

*IOException* *- on any failure, specifically if the specified path is not a folder or doesn't exist.*

**APSFile getFile(String path)**

Returns the file or folder of the specified path.

*Parameters*

*path* *- The path of the file.*

**APSDirectory getRootDirectory()**

Returns the root directory.

}

---

public *interface* **APSFilesystemService** [se.natusoft.osgi.aps.api.core.filesystem.service] {

This provides a filesystem for use by services/applications. Each filesystem has its own root that cannot be navigated outside of.

Services or application using this should do something like this in their activators:

```
APSFilesystemService fss;
APSFilesystem fs;

if (fss.hasFilesystem("my.file.system")) {
    fs = fss.getFilsystem("my.file.system");
}
else {
    fs = fss.createFilesystem("my.file.system");
}
```

**APSFilesystem createFilesystem(String owner) throws IOException**

Creates a new filesystem for use by an application or service. Where on disk this filesystem resides is irrelevant. It is accessed using the "owner", and will exist until it is removed.

*Parameters*

*owner* *- The owner of the filesystem or rather a unique identifier of it. Consider using application or service*

*package.*

*Throws*

*IOException - on any failure. An already existing filesystem for the "owner" will cause this exception.*

**boolean hasFilesystem(String owner)**

Returns true if the specified owner has a filesystem.

*Parameters*

*owner - The owner of the filesystem or rather a unique identifier of it.*

**APSFilesystem getFilesystem(String owner) throws IOException**

Returns the filesystem for the specified owner.

*Parameters*

*owner - The owner of the filesystem or rather a unique identifier of it.*

*Throws*

*IOException - on any failure.*

**void deleteFilesystem(String owner) throws IOException**

Removes the filesystem and all files in it.

*Parameters*

*owner - The owner of the filesystem to delete.*

*Throws*

*IOException - on any failure.*

}

---