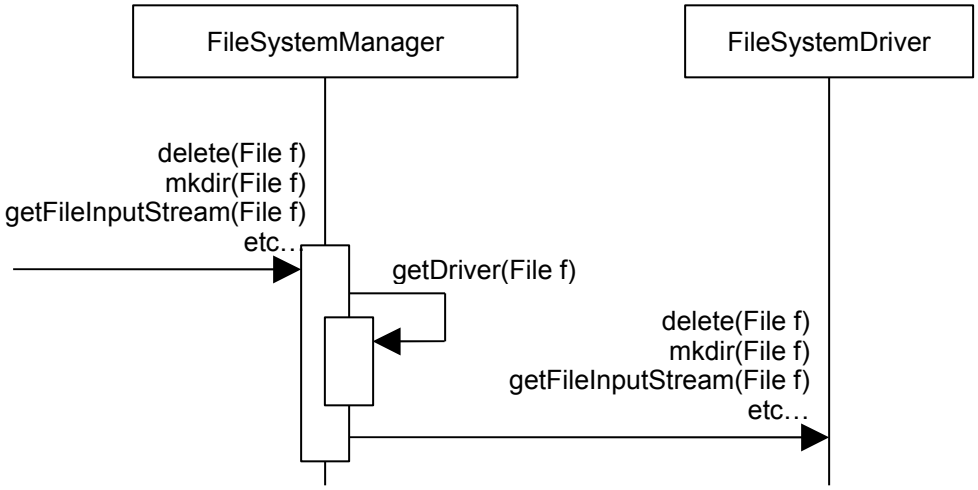
Areca's file-system access layer

The java.io.File class is simply used as a pointer by Areca. That means that the read/write methods (such as delete, mkdir, isFile, …) are NEVER invoked directly.
Instead, Areca accesses the file system by calling specific classes that implement the « FileSystemDriver » interface. These drivers can be registered in the FileSystemManager by invoking the « registerDriver(File, FileSystemDriver » method.

Each access to the file system follows the following procedure :

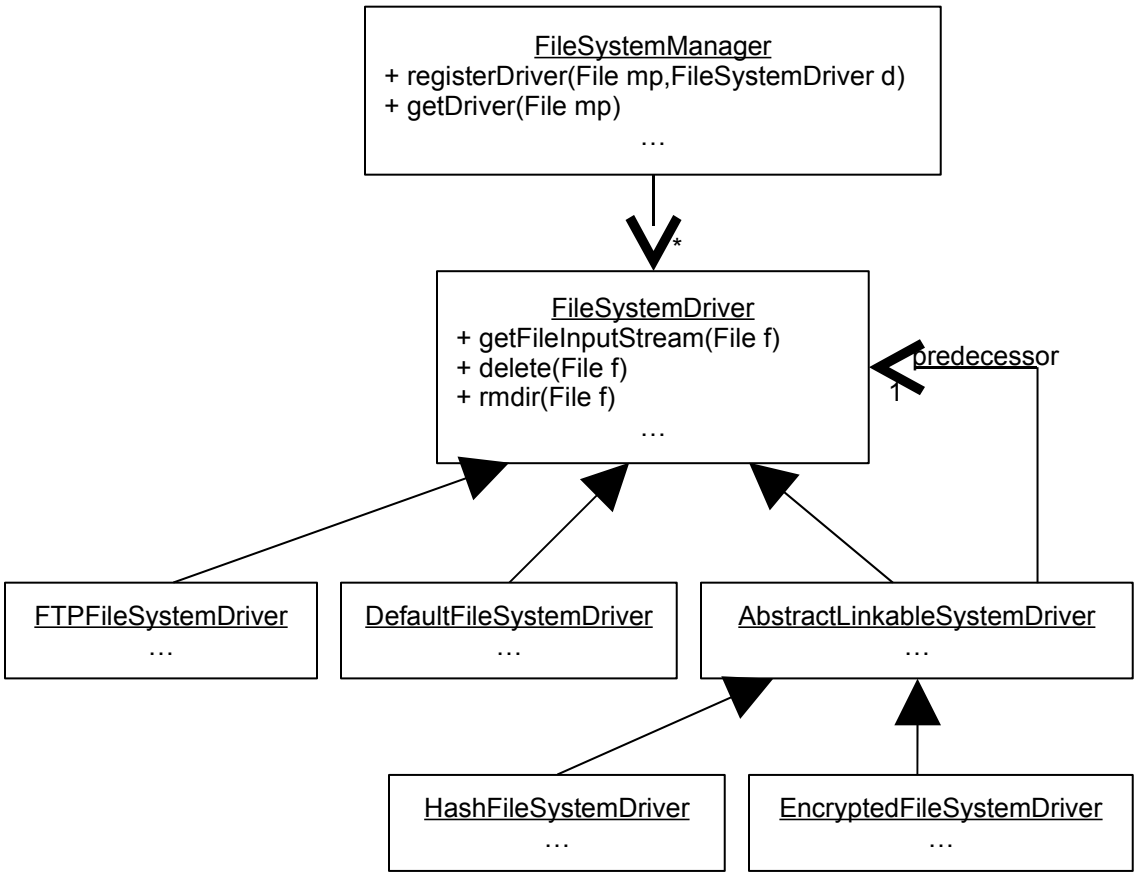


FileSystemDriver class hierarchy

Each FileSystemDriver implements a specific file-access behaviour.

The main implementations are :

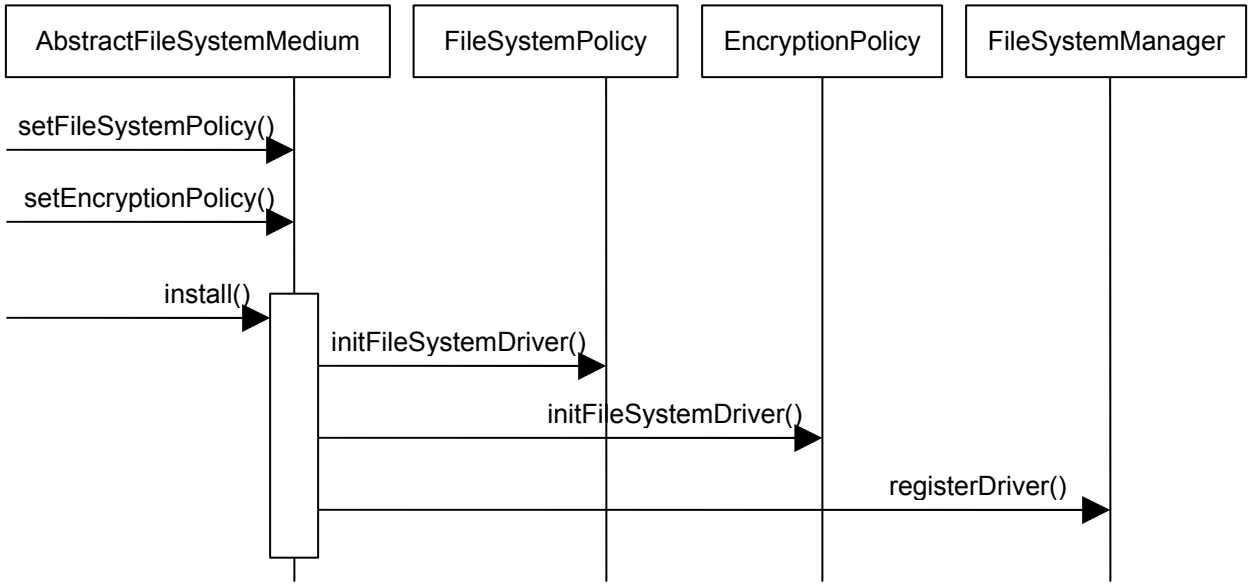
- The DefaultFileSystemDriver, which routes methods invocations to the corresponding methods of the « java.io.File » class (for instance delete, mkdir, …)
- The FTPFileSystemDriver, which routes all filesystem calls to a FTP server
- The AbstractLinkableFilesystemDriver, which wrapps another FilesystemDriver (the « predecessor » - for instance a DefaultFileSystemDriver) and implements a decorator pattern : it triggers specific behaviour (for instance encryption) before calling its predecessor.



Target storage configuration

Each Target references a Medium which is used for file storage operations (the main implementation is the AbstractIncrementalArchiveMedium). Once the target is instanciated, and its medium properly configured, the « install » method is called by Areca. This method calls the initFileSystemDriver of the FileSystemPolicy and the EncryptionPolicy classes.

These methods instanciate and configure a specific FileSystemDriver, which is registered in the FileSystemManager. Further calls to this FileSystemManager will use the appropriate driver and activate encryption / ftp accesses / … behaviours.



The Plugin API

Areca provides a plugin API which allows developers to implement their own storage policy. Storage policies must be implemented as subclasses of the FileSystemPolicy interface. Once this policy is instanciated, a corresponding FilesystemDriver can be created, configured and registered as described above.

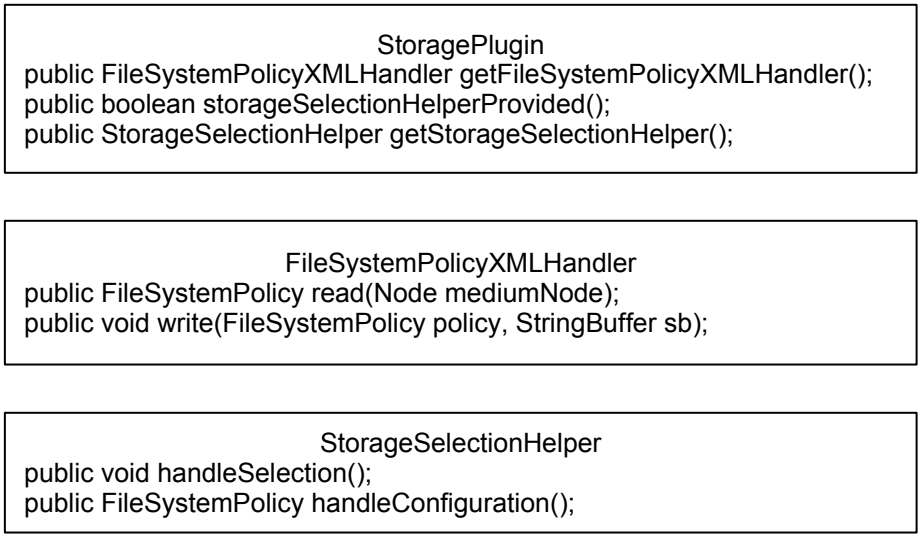
Areca may instanciate a new FileSystemPolicy in two cases :

- On startup, when the target's XML configuration is read : A target is created, and its corresponding Medium is instanciated with its own storage policy.
- When the target configuration window is displayed (when the user creates or modifies an existing target)

That's why each new FileSystemPolicy must provide its own :

- XML adapter (implemented as a subclass of FileSystemPolicyXMLHandler), which is used to read/write the target's XML configuration
- User interface helper (implemented as a subclass of StorageSelectionHelper), which is used to display the appropriate configuration window to the user)

Both of these classes are wrapped by a StoragePlugin :



Storage plugin packaging

Each plugin must be installed in the /plugins subdirectory of Areca. Let's consider a new storage plugin called « dummy ».

A new subdirectory called « plugins /dummy » must be created, which must contain a « dummy.properties » file. This file is a deployment descriptor and contains two important informations :

- plugin.jar.file : specifies the jar file that contains the plugin implementation.
- plugin.class : specifies the plugin main class.

For instance :

```
plugin.jar.file=my_dummy_implementation.jar
plugin.class=com.application.areca.dummyplugin.DummyStoragePlugin
```

As defined in this deployment descriptor example, the /plugins/dummy directory must also contain a « my_dummy_implementation.jar » file that contains all java classes required by the plugin. This jar file must contain a class named « com.application.areca.dummyplugin.DummyStoragePlugin » which defines the plugin implementation.

Examples

Areca contains an internal plugin which is used for the FTP / FTPs storage. it can be used as an example if you want to implement your own storage plugin.
Have a look at the com.application.areca.plugins.FTPStoragePlugin class.

You can also have a look at :

- com.application.areca.plugins.StoragePluginRegistry
- com.application.areca.plugins.StoragePlugin
- com.application.areca.launcher.gui.FTPStorageSelectionHelper
- com.application.areca.adapters.FTPFileSystemPolicyXMLHandler
- com.application.areca.launcher.gui.FTPEditionWindow
- com.application.areca.launcher.gui.TargetEditionWindow (particularly the « initGeneralTab » method)