JCIFS

JCIFS Component

Available as of Camel 2.11.0

This component provides access to remote file systems over the CIFS/SMB networking protocol. The **camel-jcifs** library is provided by the Camel Extra project which hosts all *GPL related components for Camel.

Maven users will need to add the following dependency to their `pom.xml` for this component:

xml <dependency> <groupId>org.apache-extras.camel-extra</groupId> <artifactId>camel-jcifs</artifactId> <version>x.x.x</version> <!-- use the same version as your Camel core version --> </dependency> Consuming from remote server
Make sure you read the section titled *Default when consuming files* further below for details related to consuming files.

URI format

smb://[[[domain;]username[:password]@]server[:port]/[[share/[dir/]]]][?options]

Where **share** represents the share to connect to and **dir** is optionaly any underlying directory. Can contain nested folders.

You can append query options to the URI in the following format, `?option=value&option=value&...`

This component uses the JCIFS library for the actual CIFS/SMB work.

URI Options

The options below are exclusive for the JCIFS component.

confluenceTableSmall

Name	Default Value	Description
`password`	`null`	**Mandatory** Specifies the password to use to log in to the remote file system.
`localWorkDirectory`	`null`	When consuming, a local work directory can be used to store the remote file content directly in local files, to avoid loading the content into memory. This is beneficial, if you consume a very big remote file and thus can conserve memory. See below for more details.

More options
See File for more options as all the options from File is inherited.

For example to set the `localWorkDirectory` to "/tmp" you can do:

java from("smb://foo@myserver.example.com/sharename?password=secret&localWorkDirectory=/tmp") .to("bean:foo");

You can have as many of these options as you like.

Examples

```
smb://foo@myserver.example.com/sharename?password=secret
smb://companydomain;foo@myserver.company.com/sharename?password=secret
```

More information
This component is an extension of the File component. So there are more samples and details on the File component page.

Message Headers

The following message headers can be used to affect the behavior of the component

confluenceTableSmall

Header	Description
`CamelFileName`	Specifies the output file name (relative to the endpoint directory) to be used for the output message when sending to the endpoint. If this is not present and no expression either, then a generated message ID is used as the filename instead.
`CamelFileNameProduced`	The actual absolute filepath (path + name) for the output file that was written. This header is set by Camel and its purpose is providing end-users the name of the file that was written.
`CamelFileBatchIndex`	Current index out of total number of files being consumed in this batch.
`CamelFileBatchSize`	Total number of files being consumed in this batch.
`CamelFileHost`	The remote hostname.

`CamelFileLocalWorkPath`	Path to the local work file, if local work directory is used.

Using Local Work Directory

Camel JCIFS supports consuming from remote servers and downloading the files directly into a local work directory. This avoids reading the entire remote file content into memory as it is streamed directly into the local file using `FileOutputStream`.

Camel JCIFS will store to a local file with the same name as the remote file. And finally, when the Exchange is complete the local file is deleted.

So if you want to download files from a remote server and store it as files then you need to route to a file endpoint such as:

java from("smb://foo@myserver.example.com/sharename?password=secret&localWorkDirectory=/tmp") .to("file://inbox"); Optimization by renaming work file
The route above is ultra efficient as it avoids reading the entire file content into memory. It will download the remote file directly to a local file stream. The `java.io.File` handle is then used as the Exchange body. The file producer leverages this fact and can work directly on the work file `java.io.File` handle and perform a `java.io.File.rename` to the target filename. As Camel knows it's a local work file, it can optimize and use a rename instead of a file copy, as the work file is meant to be deleted anyway.

Samples

In the sample below we set up Camel to download all the reports from the SMB/CIFS server once every hour (60 min) and store it as files on the local file system.

java protected RouteBuilder createRouteBuilder() throws Exception { return new RouteBuilder() { public void configure() throws Exception { // we use a delay of 60 minutes (eg. once pr. hour) we poll the server long delay = 60 * 60 * 1000L; // from the given server we poll (= download) all the files // from the public/reports folder and store this as files // in a local directory. Camel will use the filenames from the server from("smb://foo@myserver.example.com /public/reports?password=secret&delay=" + delay) .to("file://target/test-reports"); } }; } from("smb://foo@myserver.example.com/sharename? password=secret&delay=60000") .to("file://target/test-reports")

And the route using Spring DSL:

xml <route> <from uri="smb://foo@myserver.example.com/sharename?password=secret&delay=60000"/> <to uri="file://target/test-reports"/> </route>

Filter using `org.apache.camel.component.file.GenericFileFilter`

Camel supports pluggable filtering strategies. This strategy it to use the build in `org.apache.camel.component.file.GenericFileFilter` in Java. You can then configure the endpoint with such a filter to skip certain filters before being processed.

In the sample we have built our own filter that only accepts files starting with report in the filename.

{snippet:id=e1|lang=java|url=camel/trunk/components/camel-ftp/src/test/java/org/apache/camel/component/file/remote/FromFtpRemoteFileFilterTest.java}

And then we can configure our route using the **filter** attribute to reference our filter (using # notation) that we have defined in the spring XML file:

xml <!-- define our sorter as a plain spring bean --> <bean id="myFilter" class="com.mycompany.MyFileFilter"/> <route> <from uri="smb://foo@myserver. example.com/sharename?password=secret&filter=#myFilter"/> <to uri="bean:processInbox"/> </route>

Filtering using ANT path matcher

The ANT path matcher is a filter that is shipped out-of-the-box in the **camel-spring** jar. So you need to depend on **camel-spring** if you are using Maven. The reason is that we leverage Spring's AntPathMatcher to do the actual matching.

The file paths are matched with the following rules:

- ? matches one character
- * matches zero or more characters
- * * matches zero or more directories in a path

The sample below demonstrates how to use it:

xml <camelContext xmlns="http://camel.apache.org/schema/spring"> <template id="camelTemplate"/> <!-- use myFilter as filter to allow setting ANT paths for which files to scan for --> <endpoint id="mySMBEndpoint" uri="smb://foo@myserver.example.com/sharename?password=secret& recursive=true&filter=#myAntFilter"/> <route> <from ref="mySMBEndpoint"/> <to uri="mock:result"/> </route> </camelContext> <!-- we use the AntPathMatcherRemoteFileFilter to use ant paths for includes and exclude --> <bean id="myAntFilter" class="org.apache.camel.component.file. AntPathMatcherGenericFileFilter"> <!-- include any files in the sub folder that has day in the name --> <property name="includes" value="**/subfolder/** /*day*"/> <!-- exclude all files with bad in name or .xml files. Use comma to separate multiple excludes --> <property name="excludes" value="**/*bad*,**/*. xml"/> </bean> Endpoint See Also

- File2