File Language

File Expression Language

> **File language is now merged with Simple language**
> From **Camel 2.2**: the file language is now merged with Simple language which means you can use all the file syntax directly within the simple language.

The File Expression Language is an extension to the Simple language, adding file related capabilities. These capabilities are related to common use cases working with file path and names. The goal is to allow expressions to be used with the File and FTP components for setting dynamic file patterns for both consumer and producer.

Syntax

This language is an **extension** to the Simple language so the Simple syntax applies also. So the table below only lists the additional. By contrast to the Simple language, the File Language also supports the use of Constant expressions to enter a fixed filename, for example.

All the file tokens use the same expression name as the method on the **`java.io.File`** object. For example: **`file:absolute`** refers to the **`java.io.File.getAbsolute()`** method.

> **Note**: not all expressions are supported by the current Exchange. For example, the FTP component supports some of the options, whereas the File component supports all of them.

Expression	Type	File Consumer	File Producer	FTP Consumer	FTP Producer	Description
`date:`*`command`*`:`*`pattern`*	`String`	`yes`	`yes`	`yes`	`yes`	For date formatting using the **`java.text.SimpleDateFormat`** patterns which is an **extension** to the Simple language. Additional command is: **`file`** (consumers only) for the last modified timestamp of the file. **Note**: all the commands from the Simple language can also be used.
`file:absolute`	`Boolean`	`yes`	`no`	`no`	`no`	Refers to whether the file is regarded as absolute or relative.
`file:absolute.path`	`String`	`yes`	`no`	`no`	`no`	Refers to the absolute file path.
`file:ext`	`String`	`yes`	`no`	`yes`	`no`	Refers to the file extension only.
`file:length`	`Long`	`yes`	`no`	`yes`	`no`	Refers to the file length returned as a **`Long`** type.
`file:modified`	`Date`	`yes`	`no`	`yes`	`no`	Refers to the file last modified returned as a **`Date`** type.
`file:name`	`String`	`yes`	`no`	`yes`	`no`	Refers to the file name (is relative to the starting directory, see note below).
`file:name.ext`	`String`	`yes`	`no`	`yes`	`no`	**Camel 2.3:** refers to the file extension only.
`file:name.ext.single`	`String`	`yes`	`no`	`yes`	`no`	**Camel 2.14.4/2.15.3:** refers to the file extension. If the file extension has multiple dots, then this expression strips and only returns the last part.
`file:name.noext`	`String`	`yes`	`no`	`yes`	`no`	Refers to the file name with no extension (is relative to the starting directory, see note below).
`file:name.noext.single`	`String`	`yes`	`no`	`yes`	`no`	**Camel 2.14.4/2.15.3:** refers to the file name with no extension (is relative to the starting directory, see note below). If the file extension has multiple dots, then this expression strips only the last part, and keep the others.
`file:onlyname`	`String`	`yes`	`no`	`yes`	`no`	Refers to the file name only with no leading paths.
`file:onlyname.noext`	`String`	`yes`	`no`	`yes`	`no`	Refers to the file name only with no extension and with no leading paths.

`file:onlyname.noext.single`	`String`	`yes`	`no`	`yes`	`no`	**Camel 2.14.4/2.15.3:** refers to the file name only with no extension and with no leading paths. If the file extension has multiple dots, then this expression strips only the last part, and keep the others.
`file:parent`	`String`	`yes`	`no`	`yes`	`no`	Refers to the file parent.
`file:path`	`String`	`yes`	`no`	`yes`	`no`	Refers to the file path.
`file:size`	`Long`	`yes`	`no`	`yes`	`no`	**Camel 2.5:** refers to the file length returned as a **`Long`** type.

File Token Example

Relative Paths

We have a **`java.io.File`** handle for the file **`hello.txt`** in the following *relative* directory: **`.\filelanguage\test`**. And we configure our endpoint to use this starting directory **`.\filelanguage`**.

The file tokens returned are:

Expression	Returns
`file:absolute`	`false`
`file:absolute.path`	`\workspace\camel\camel-core\target\filelanguage\test\hello.txt`
`file:ext`	`txt`
`file:name`	`test\hello.txt`
`file:name.ext`	`txt`
`file:name.noext`	`test\hello`
`file:onlyname`	`hello.txt`
`file:onlyname.noext`	`hello`
`file:parent`	`filelanguage\test`
`file:path`	`filelanguage\test\hello.txt`

Absolute Paths

We have a **`java.io.File`** handle for the file **`hello.txt`** in the following *absolute* directory: **`\workspace\camel\camel-core\target\filelanguage\test`**. And we configure out endpoint to use the absolute starting directory: **`\workspace\camel\camel-core\target\filelanguage`**.

The file tokens return are:

Expression	Returns
`file:absolute`	`true`
`file:absolute.path`	`\workspace\camel\camel-core\target\filelanguage\test\hello.txt`
`file:ext`	`txt`
`file:name`	`test\hello.txt`
`file:name.ext`	`txt`
`file:name.noext`	`test\hello`
`file:onlyname`	`hello.txt`
`file:onlyname.noext`	`hello`

`file:parent`	`\workspace\camel\camel-core\target\filelanguage\test`
`file:path`	`\workspace\camel\camel-core\target\filelanguage\test\hello.txt`

Examples

You can enter a fixed Constant expression such as **`myfile.txt`**:

```
fileName="myfile.txt"
```

Lets assume we use the file consumer to read files and want to move the read files to backup folder with the current date as a sub folder. This can be achieved using an expression like:

```
fileName="backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

relative folder names are also supported so suppose the backup folder should be a sibling folder then you can append **`..`** as:

```
fileName="../backup/${date:now:yyyyMMdd}/${file:name.noext}.bak"
```

As this is an extension to the Simple language we have access to all the goodies from this language also, so in this use case we want to use the **`in.header.type`** as a parameter in the dynamic expression:

```
fileName="../backup/${date:now:yyyyMMdd}/type-${in.header.type}/backup-of-${file:name.
noext}.bak"
```

If you have a custom **`Date`** you want to use in the expression then Camel supports retrieving dates from the message header.

```
fileName="orders/order-${in.header.customerId}-${date:in.header.orderDate:yyyyMMdd}.xm
l"
```

And finally we can also use a bean expression to invoke a POJO class that generates some **`String`** output (or convertible to **`String`**) to be used:

```
fileName="uniquefile-${bean:myguidgenerator.generateid}.txt"
```

And of course all this can be combined in one expression where you can use the File Language, Simple and the Bean language in one combined expression. This is pretty powerful for those common file path patterns.

Using Spring's `PropertyPlaceholderConfigurer` with the File Component

In Camel you can use the File Language directly from the Simple language which makes a Content Based Router easier to do in Spring XML, where we can route based on file extensions as shown below:

```
<from uri="file://input/orders"/>
  <choice>
    <when>
      <simple>${file:ext} == 'txt'</simple>
      <to uri="bean:orderService?method=handleTextFiles"/>
    </when>
    <when>
      <simple>${file:ext} == 'xml'</simple>
      <to uri="bean:orderService?method=handleXmlFiles"/>
    </when>
    <otherwise>
      <to uri="bean:orderService?method=handleOtherFiles"/>
    </otherwise>
  </choice>
```

If you use the **`fileName`** option on the File endpoint to set a dynamic filename using the File Language then make sure you use the alternative syntax (available from Camel 2.5) to avoid clashing with Spring's **`PropertyPlaceholderConfigurer`**.

bundle-context.xml

```
<bean id="propertyPlaceholder"
class="org.springframework.beans.factory.config.PropertyPlaceholderConfigurer">
  <property name="location" value="classpath:bundle-context.cfg"/>
</bean>

<bean id="sampleRoute" class="SampleRoute">
  <property name="fromEndpoint" value="${fromEndpoint}"/>
  <property name="toEndpoint" value="${toEndpoint}"/>
</bean>
```

bundle-context.cfg

```
fromEndpoint=activemq:queue:test
toEndpoint=file://fileRoute/out?fileName=test-$simple{date:now:yyyyMMdd}.txt
```

Notice how we use the **`$simple{}`** syntax in the **`toEndpoint`** above. If you don't do this, they will clash and Spring will throw an exception:

```
org.springframework.beans.factory.BeanDefinitionStoreException:
Invalid bean definition with name 'sampleRoute' defined in class path resource
[bundle-context.xml]:
Could not resolve placeholder 'date:now:yyyyMMdd'
```

Dependencies

The File language is part of **`camel-core`**.