File Exist Trigger

Fires when specified files exist. The file exist trigger periodically scans, as defined by the polling delay property, the appropriate file system and returns all files that exist and match the file criteria.

By default, the File Exist Trigger will scan indefinitely (at an interval defined by the polling delay) until the file (or files) that it is searching for appears. If, instead, you would like the trigger to wait a certain amount of time before assuming that the file will not exist, you can set a timeout to specify how long the trigger should wait for the file to appear. If a timeout is set, the trigger will follow normal timeout behavior when the timeout window elapses and the file (or files) has still not appeared.

Active Window

Defines the times when this file trigger may scan for files. If the active window is null, this file trigger may always scan for files.

Stable Period

Indicates how long a file must be stable on the file system before the trigger can pick it up. This prevents the trigger from picking up a file (for example) while another system is copying or modifying it. Defaults to "+0s", meaning that there is no stable period by default.

To use the stable period with a remote file host (like an FTP server), you will need to be sure that the time clocks are synchronized between the Flux engine and the remote file host. If you are having trouble with the stable period, double-check that the clocks on both systems are synchronized as expected. See File Triggers and Actions for more information.

Polling Delay

Sets the delay that occurs between file polling. Defaults to "+3s", 3 seconds.

File Exist Trigger and Hidden Files

If a File Exist Trigger is configured to search for files on an FTP, FTPS, or SFTP host, the files must be available to be listed with the "LS" command. If a file cannot be listed this way on the server (that is, the file is hidden), you must use the FTP Command Action to directly issue FTP commands to the server and locate the file that way.

Results

The File Exist Trigger returns its results in the flow context variable "result". These results include listings of all files that were successfully located, including source locations in various formats. You can access the results from the following fields:

Flow Context Variable	Field	Java Type	Description	Prescript / Postscript Example

RESULT	url_string_matches	List<String>	Contains URLs of files against which the file trigger matched using its file criteria. This field contains the same information as the "url_object_matches" field but in string form.
			
			This result property is a collection (even if only one file is found, the result property will be a collection containing a single value). This means that in order to access the individual URL strings returned by the File Exist Trigger, you must use a For Each Collection Element Action to step through the items one at a time.
			
			To set up the For Each Collection Element Action, just add a flow from the File Exist Trigger to the For Each Collection Element Action. Add a runtime data map to this flow with a flow context source set to "RESULT.url_string_matches" (which tells Flux "retrieve the collection stored in the url_string_matches result property"), and an action property target of "Collection". The individual URL strings can then be accessed using normal techniques for the For Each Collection Element Action.
			
			If you are confident that your collection contains only a single value, you can access that value directly using variable substitution:
			
			`${RESULT.url_string_matches.get(0)}`
			
			This will allow you to bypass the For Each Collection Element Action and directly use the URL string in a subsequent trigger or action.
				`List urlStrings = flowContext.get ("RESULT").url_string_matches;`
				
				`for (String fileUrl : urlStrings) { System.out.println("Found: " + fileUrl); }`

RESULT	url_object_matches	List<URL>	Contains URLs of files against which the file trigger matched using its file criteria. This field contains the same information as the "url_string_matches" field but in URL object form.	`List urlObjects = flowContext.get ("RESULT").url_object_matches;`
				
				`for (URL fileUrl : urlObjects) {`
				`  System.out.println("Found: " + fileUrl);`
				`}`

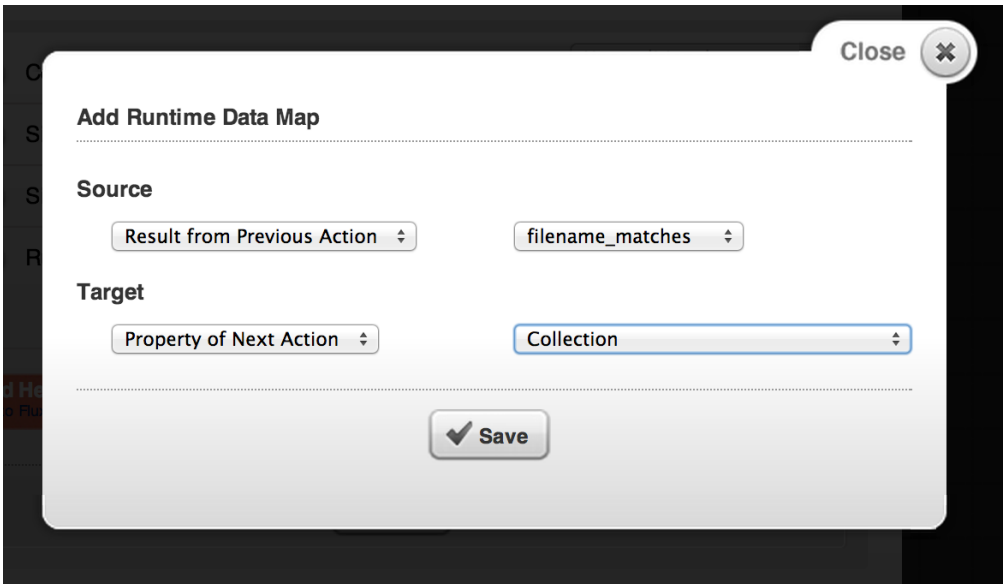
RESULT	fileinfo_matches	List<flux.file.FileInfo>	Contains details about files against which the file trigger matched using its file criteria. Details include file size, last modified date, and file permissions.
			
			This result property is a collection (even if only one file is found, the result property will be a collection containing a single value). This means that in order to access the individual File Infos returned by the File Exist Trigger, you must use a For Each Collection Element Action to step through the items one at a time.
			
			To set up the For Each Collection Element Action, just add a flow from the File Exist Trigger to the For Each Collection Element Action. Add a runtime data map to this flow with a flow context source set to "RESULT.fileinfo_matches" (which tells Flux "retrieve the collection stored in the fileinfo_matches result property"), and an action property target of "Collection".
			
			If you are confident that your collection contains only a single value, you can access that value directly using variable substitution:
			
			`${RESULT.fileinfo_matches.get(0).filename}`
			`${RESULT.fileinfo_matches.get(0).size}`
			`${RESULT.fileinfo_matches.get(0).url}`
			
			This will allow you to bypass the For Each Collection Element Action and directly use the File Info properties in a subsequent trigger or action.
			
			See "Using File Infos" below for more information on file info objects.
				`List fileInfos = flowContext.get ("RESULT").fileinfo_matches;`
				
				`for (FileInfo fileInfo : fileInfos) {`
				`  System.out.println("Found: " + fileInfo.filename);`
				`}`
				
				When pulling the info after passing it to a For Each Collection Element action use:
				
				`Complete FileInfo Object = ${i}`
				`Filename = ${i.filename}`
				`Path to file = ${i.parent}`
				`Is Readable = ${i.read?string ("yes","no")}`
				`Is Writable = ${i.read?string ("yes","no")}`
				`Last Modified = ${i.lastModified?datetime}`
				`size = ${i.size}`
				`url = ${i.url}`
				
				Output:
				
				`Complete FileInfo Object = FileInfo (read=true, write=true, filename='temp.xxx', lastModified=Sat Oct 18 18:18:47 CDT 2014, parent='c:\', size=27, url=file:/c:/temp.xxx)`
				`Filename = temp.xxx`
				`Path to file = c:\`
				`Is Readable = yes`
				`Is Writable = yes`
				`Last Modified = Oct 18, 2014 6:18:47 PM`
				`size = 27`
				`url = file:/c:/temp.xxx`

RESULT	filename_matches	List<String>	Contains simple, base file names, without path information, of files against which the file trigger matched using its file criteria.
			
			This result property is a collection (even if only one file is found, the result property will be a collection containing a single value). This means that in order to access the individual filenames returned by the File Exist Trigger, you must use a For Each Collection Element Action to step through the items one at a time.
			
			To set up the For Each Collection Element Action, just add a flow from the File Exist Trigger to the For Each Collection Element Action. Add a runtime data map to this flow with a flow context source set to "RESULT.filename_matches" (which tells Flux "retrieve the collection stored in the filename_matches result property"), and an action property target of "Collection". The individual filenames can then be accessed using normal techniques for the For Each Collection Element Action.
			
			If you are confident that your collection contains only a single value, you can access that value directly using variable substitution:
			
			`${RESULT.filename_matches.get(0)}`
			
			This will allow you to bypass the For Each Collection Element Action and directly use the filename in a subsequent trigger or action.
				`List filenames = flowContext.get ("RESULT").filename_matches;`
				
				`for (String fileUrl : filenames) {`
				`  System.out.println("Found: " + fileUrl);`
				`}`

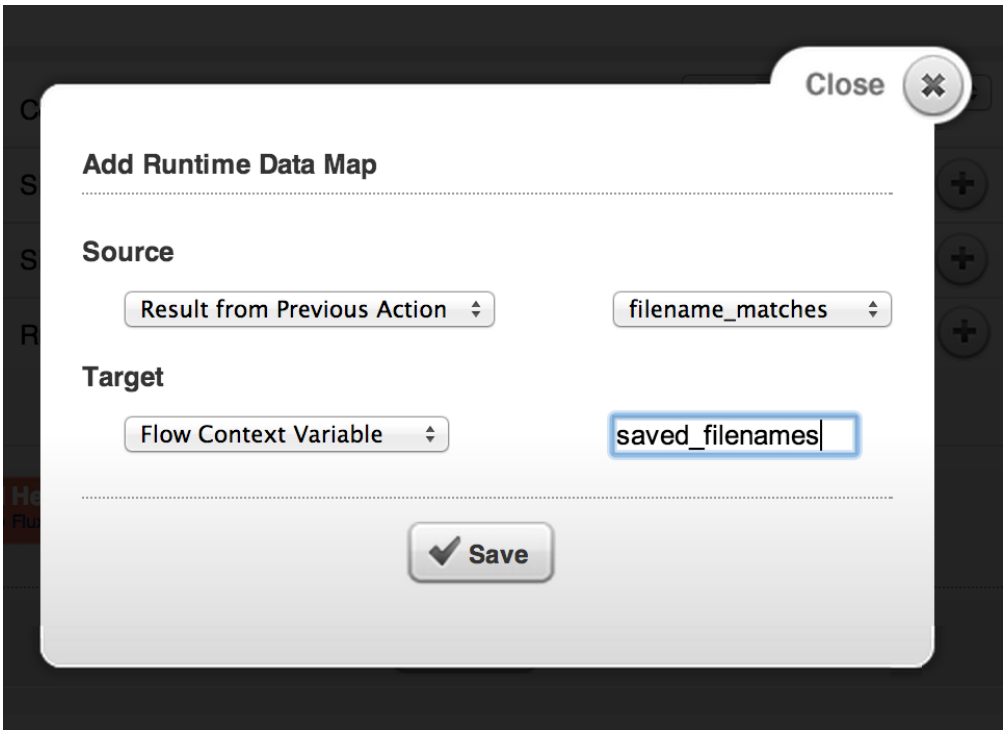
Passing Results with a Runtime Data Map

You can use a Runtime Data Map to specify one of the result fields as a collection on a For Each Collection Element Action, or to copy the result field into a new variable.

To set the collection for a For Each Collection Element Action, add the following data map on the flow going into the For Each action:



To copy the collection into a new variable (for future reference or to reuse the data later in the workflow), you could use a data map like:



Using File Infos

File Infos are complex data types (meaning that the information stored by a File Info is collected in special properties on the File Info) so the file info cannot be directly accessed using normal variable substitution. To retrieve a particular property from the File Info, you will need a separate runtime data map to store the property in its own variable, where it can then be retrieved directly.

For example, if you wanted to retrieve the "filename" property of the File Info, you could use a runtime data map (on a flow from the For Each Collection Element Action to the next trigger or action in the workflow) with a flow context source set to "<loop index>.filename" (meaning "get the filename property of the item stored as the variable <loop index>", since in this case the File Info is stored in a variable with the name of whatever the For Each Collection Element Action has set for its loop index), and a flow context target of "filename". This will retrieve the "filename" property from the File Info, then store it in a new variable named "filename" so it can be accessed using variable substitution.

You can also use multiple runtime data map source / targets to get several properties from the File Info. For example, if the For Each Collection Element Action had a loop index of "fileinfo", and you wanted to retrieve all of the properties available from the File Info, you could use a runtime data map like the following (note that both "source" and "target" should be set to use the "Flow Context" in this case):

Source	Target
fileinfo.filename	filename
fileinfo.lastModified	lastModified
fileinfo.parent	parent
fileinfo.read	read
fileinfo.size	size
fileinfo.url	url
fileinfo.write	write

You could then use variable substitution to access those properties individually: ${filename}, ${size}, etc.