Analysing Output

Action output monitors make it easy to react to the presence of a word, such as "error" in the output from an action.

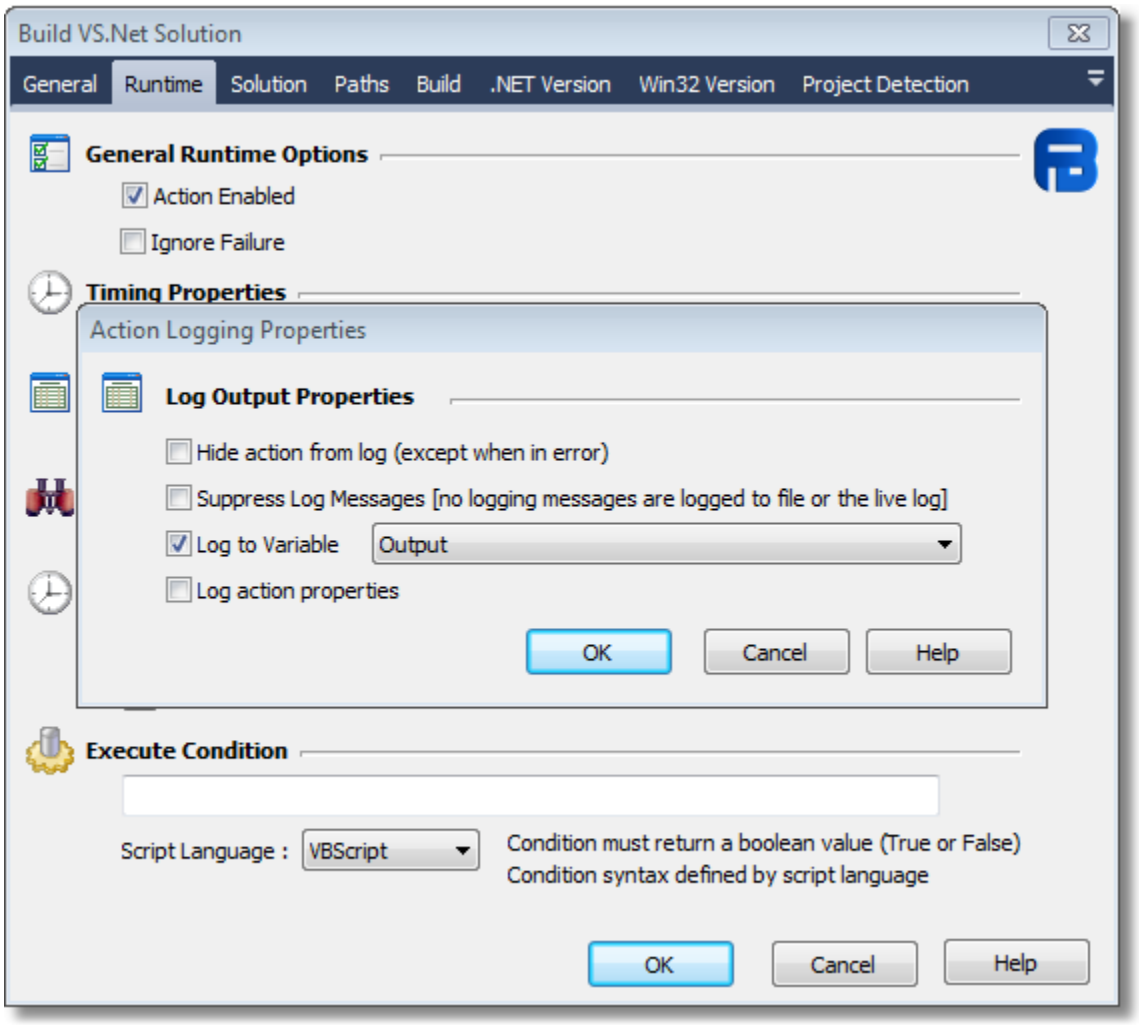
Example Problems

If you want to abort the build when there are more than five errors, or there is a requirement to process each line of the output somehow. The following are some solutions to those two problems;

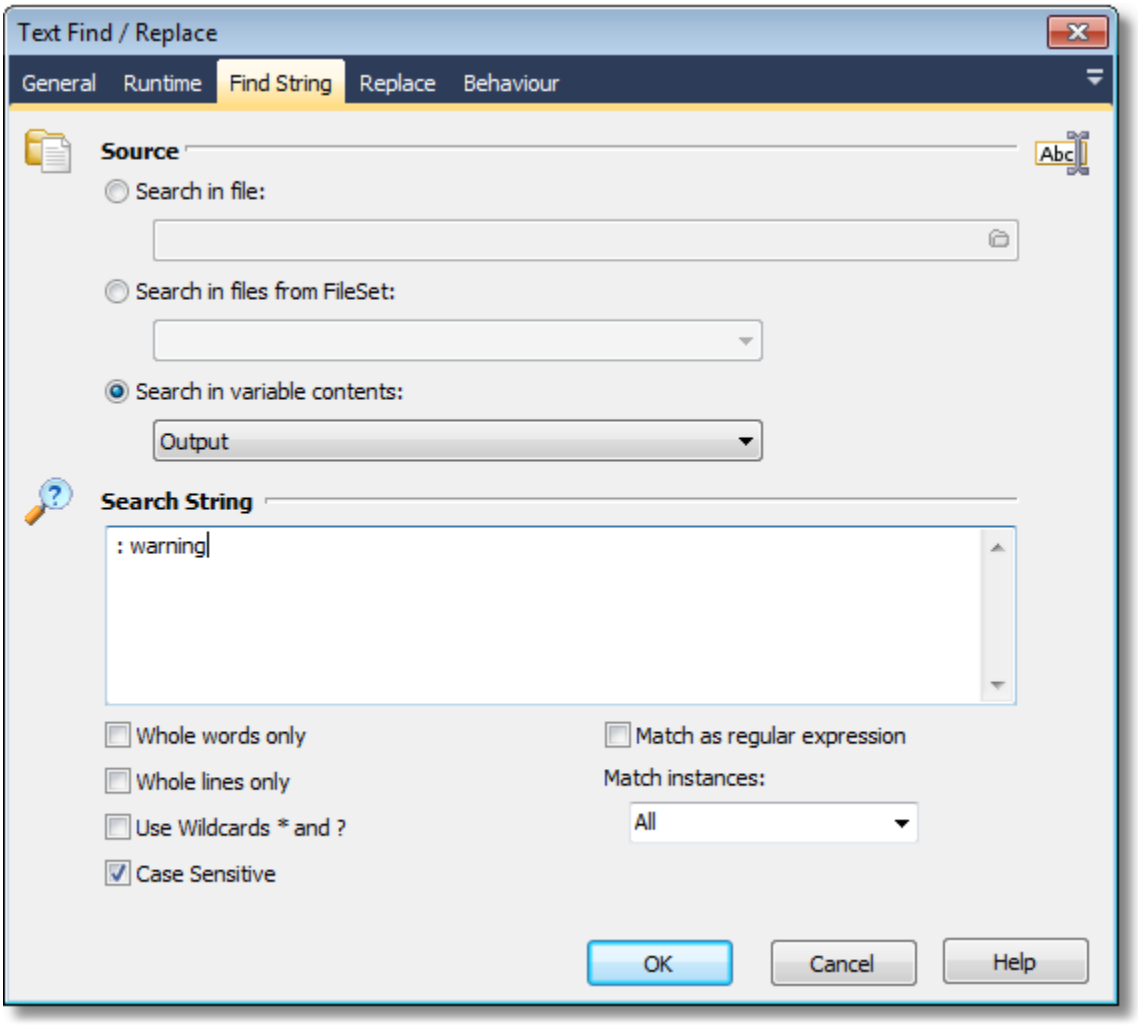
Counting warnings

The key is to log the output of the action to a variable, then analyse the contents of the variable. You'll need two variables, "Output" and "Count"

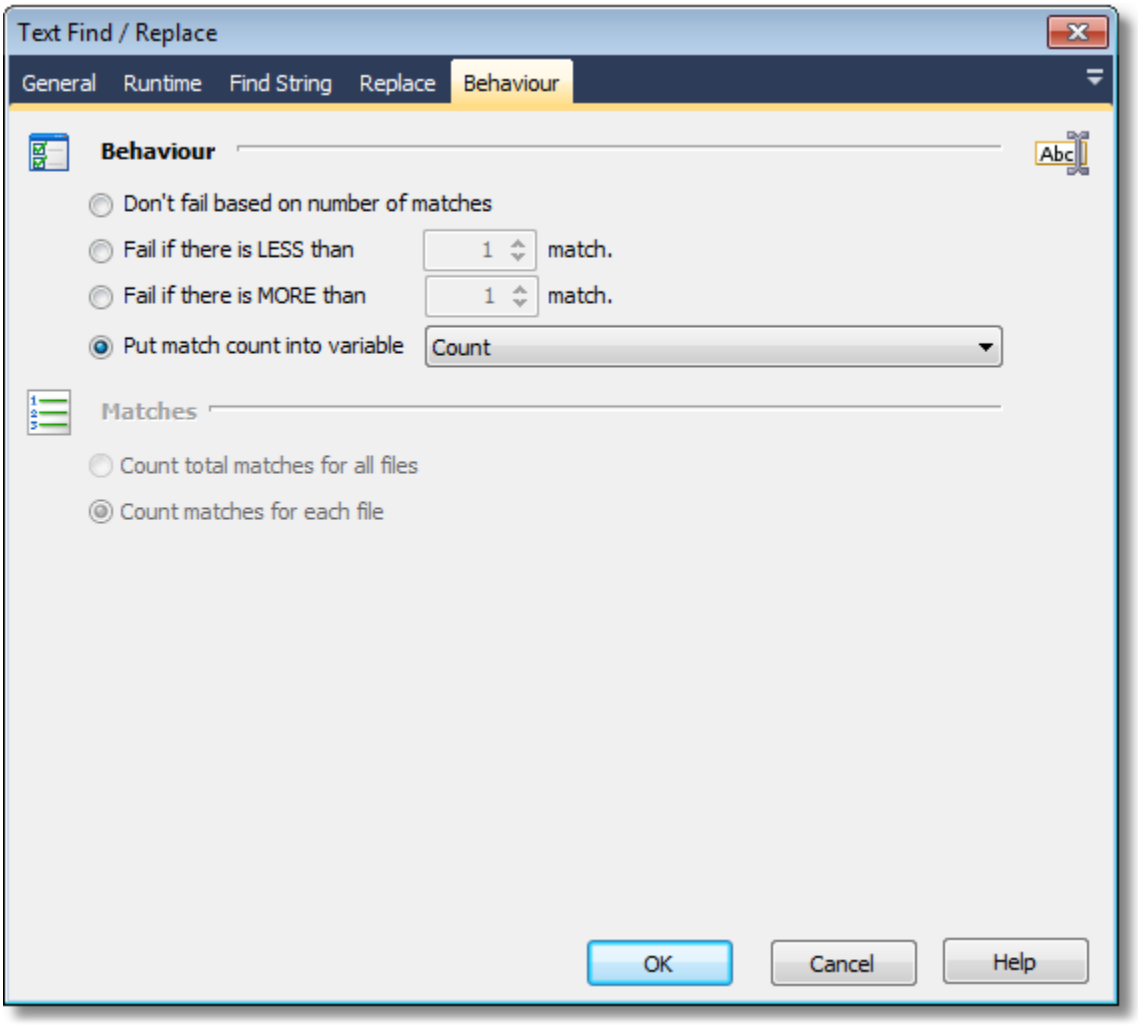
First, on the **Runtime** tab of the relevant action, go to "Logging Properties" and select "Log to Variable".



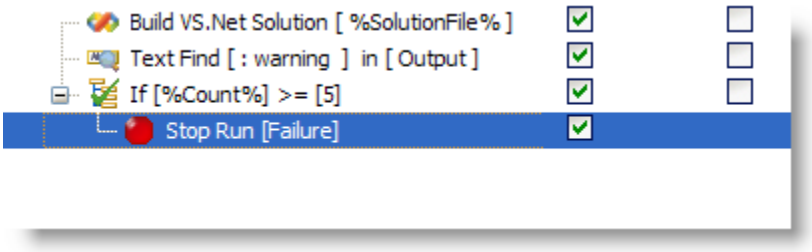
Next, use a "Text Find / Replace" action to count the number of times the string appears:



On the **Behaviour** tab, set the variable to hold the number of matches:



Now you're all set to use the variable however you like:

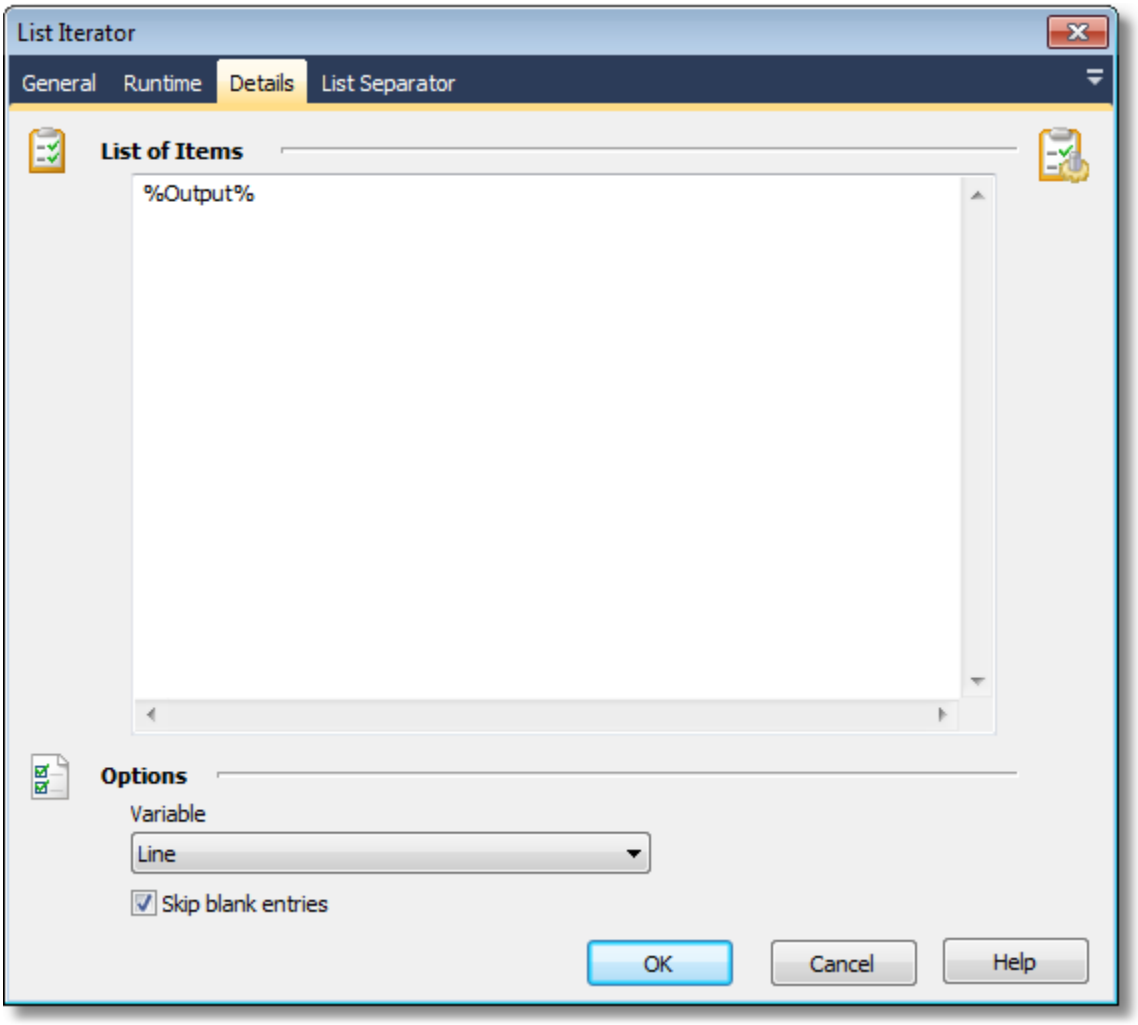


Processing a log line by line

Let's say the output from some external program is very verbose, and all you want is lines that contain "Image: " followed by a filename.

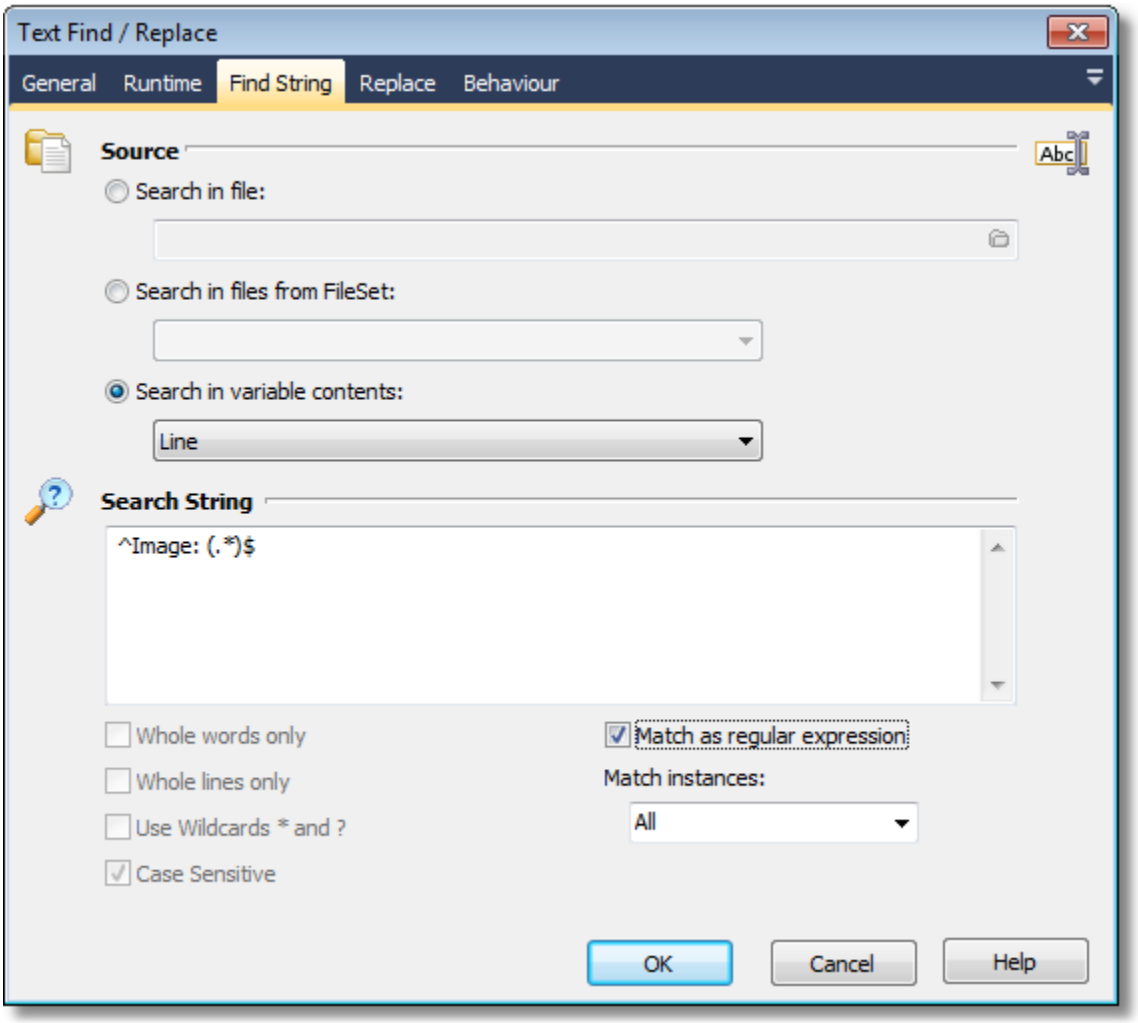
Start by logging the output of action to the Output variable. You will also need a variable to hold each line of output. Call it "Line".

Next, use a List Iterator Action action. Use %Output% as the "List of Items" value. At runtime, it will be expanded to the full value of the log. Don't worry about the size, FinalBuilder has a very large upper limit on variable size.



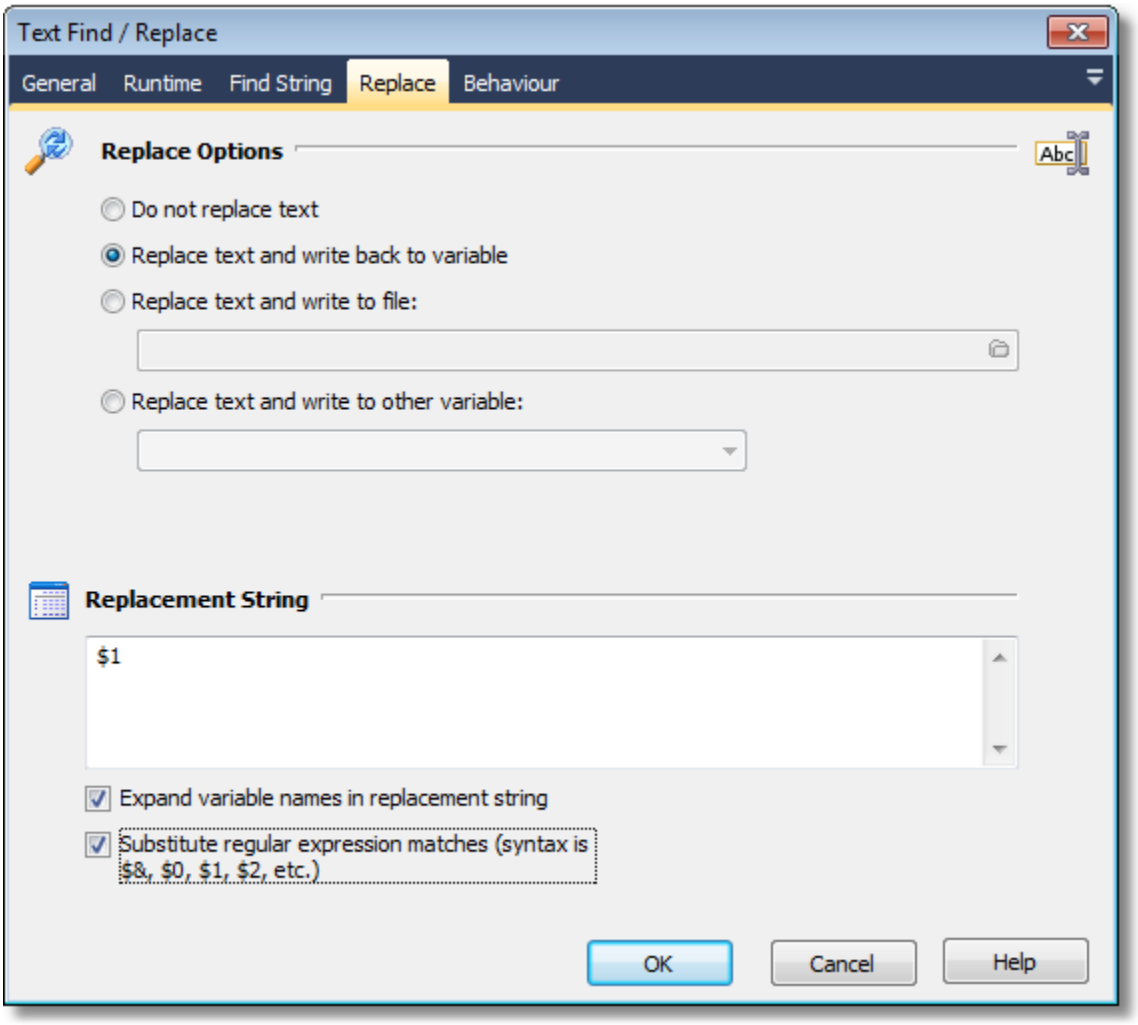
Leave the "List Separator" settings as the default: a carriage return/line feed.

Now for each line, we use a Text Find / Replace action to reduce a line containing the key string down to just the image filename itself:

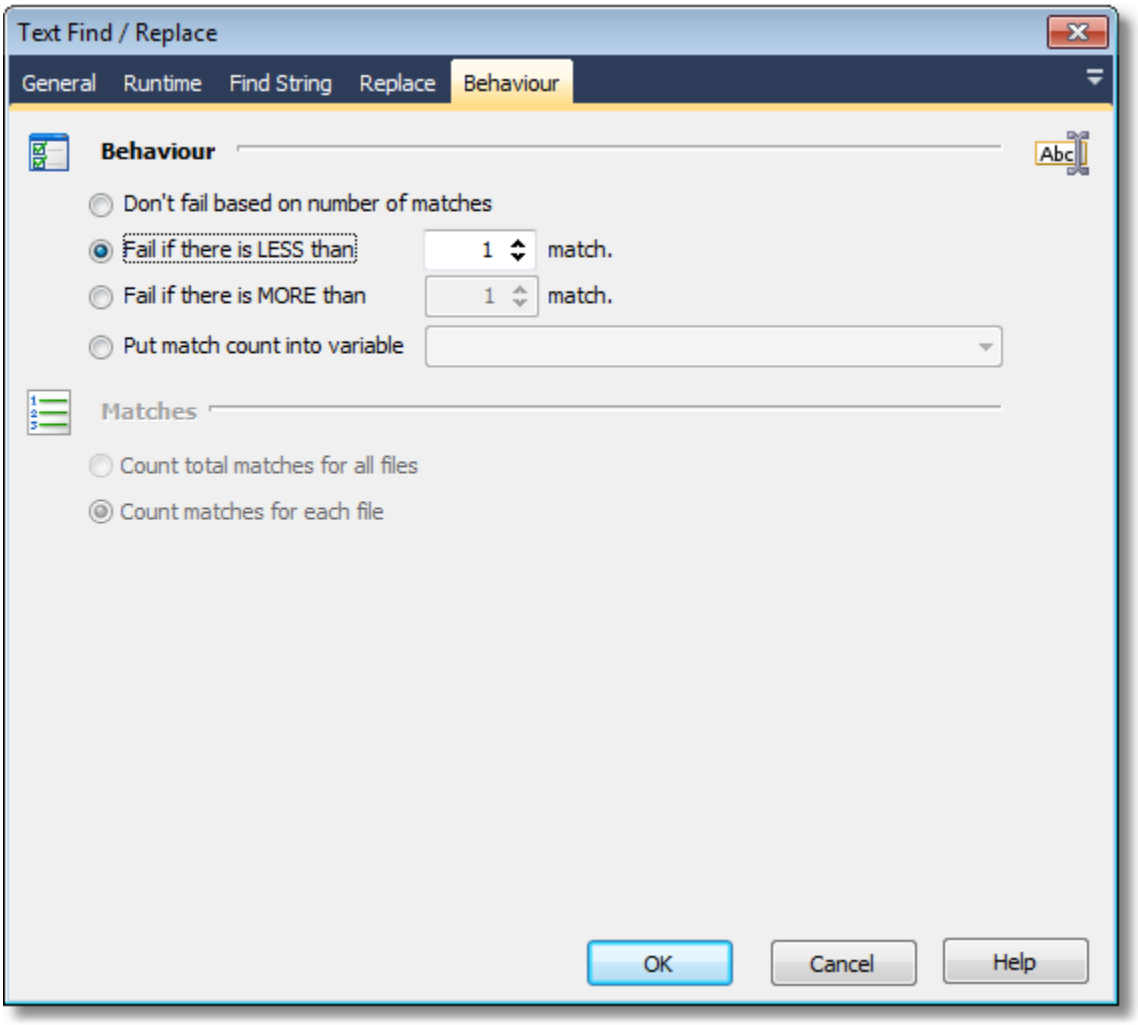


This regular expression means "beginning at the start of the line, match the word Image, a colon, a space, then store the whole rest of the line as subexpression 1".

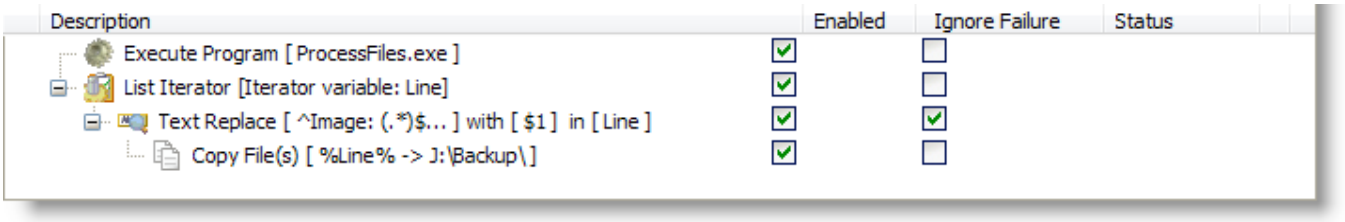
On the **Replace** tab, we write that subexpression back to the same variable:



Finally, on the **Behaviour** tab, we set the action to fail if it didn't match. We do this because we want to do further processing on a line that matches.



Now, we can add whatever processing we like. The "line" variable at this point contains just the name of the image found in the output. We set the "Text Replace" action to ignore failure. The loop should carry on for each line that doesn't have the text we're looking for.



There we have it!

Summary

1. The program runs, logging its output to a variable called "Output"
2. The list iterator cycles over that output, placing each line in a variable called "Line"
3. The Text Replace action then reduces that line down to just the image name, or fails if it's not an image line.
4. If the text is found, the file is then copied somewhere.