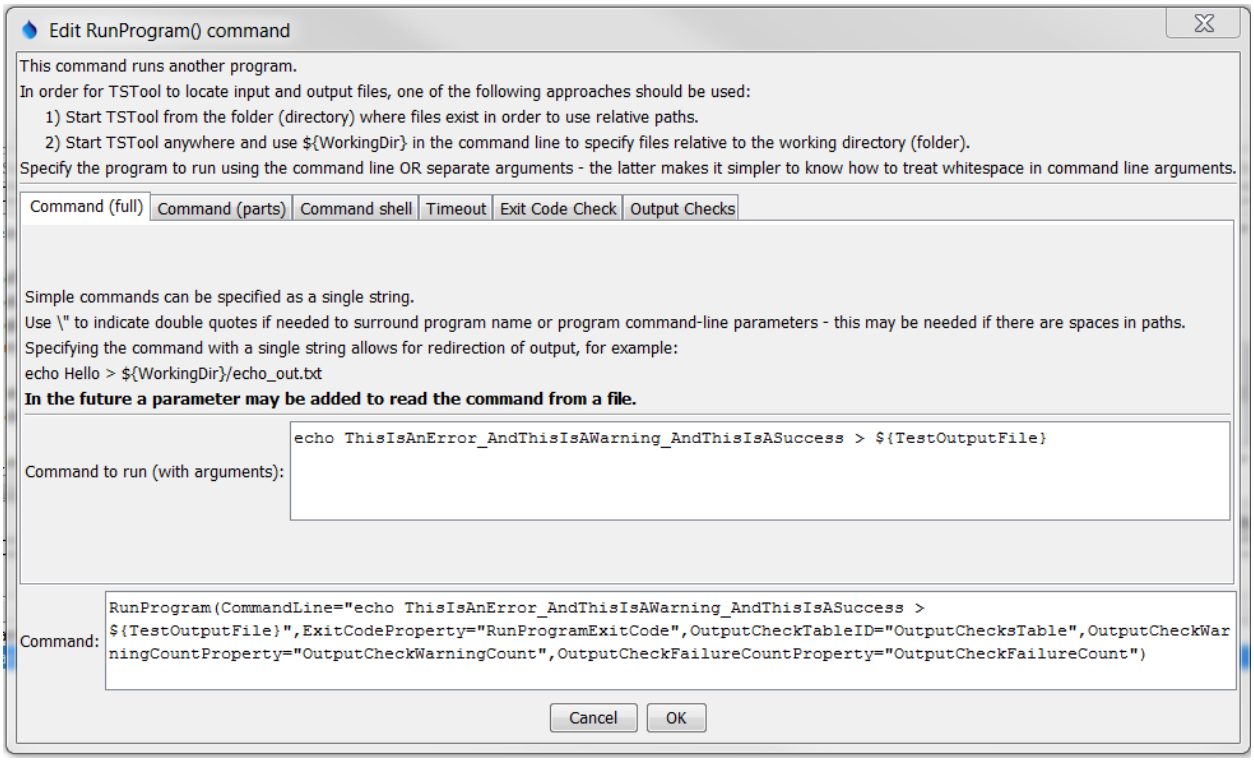
Command Reference: RunProgram()

Run an external program

Version 11.09.00, 2016-02-24

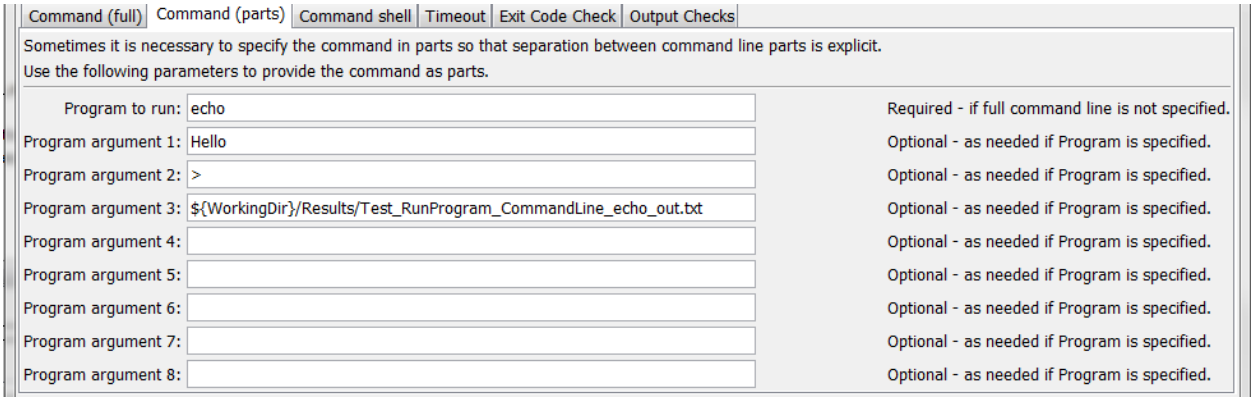
The `RunProgram()` command runs an external program, given the full command line or individual command line parts, and waits until the program is finished before processing additional commands. The TSTool command will indicate a failure if the exit status from the program being run is non-zero. It is therefore possible to call an external program that reads and/or writes recognized time series formats to perform processing that TSTool cannot. One use of this command is to create a calibration environment where a model is run and then the results are read and displayed using TSTool. It is also useful to use TSTool's testing features to implement quality control checks for other software tools.

TSTool internally maintains a working folder (directory) that is used to convert relative paths to absolute paths to locate files. The working folder is by default the location of the last command file that was opened. The external program may assume that the working folder is the location from which TSTool software was started (or the installation location if started from a menu). Therefore, it may be necessary to run TSTool in batch mode from the directory where the external software's data files exist, use absolute paths to files, or use the `${WorkingDir}` property in the command line. Use `\"` in the command line or arguments to surround whitespace. Some operating systems may have limitations on command line length. The following dialog is used to edit the command and illustrates the command syntax.



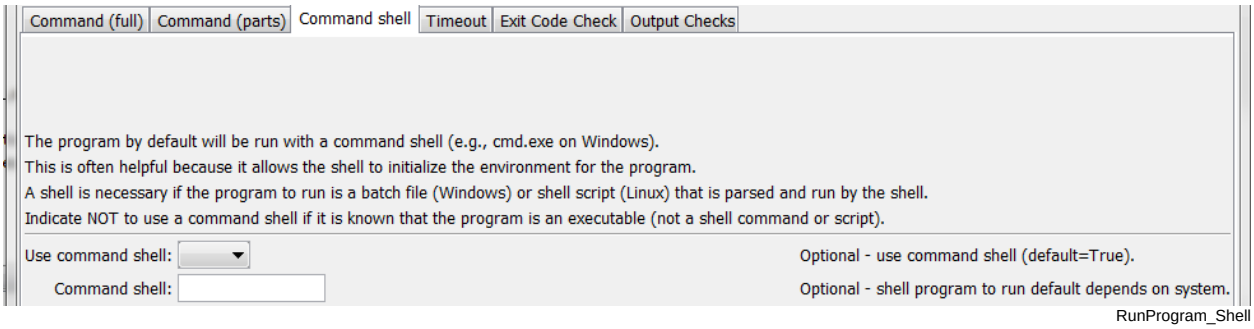
RunProgram

RunProgram() Command Editor when Specifying Command Line in Full



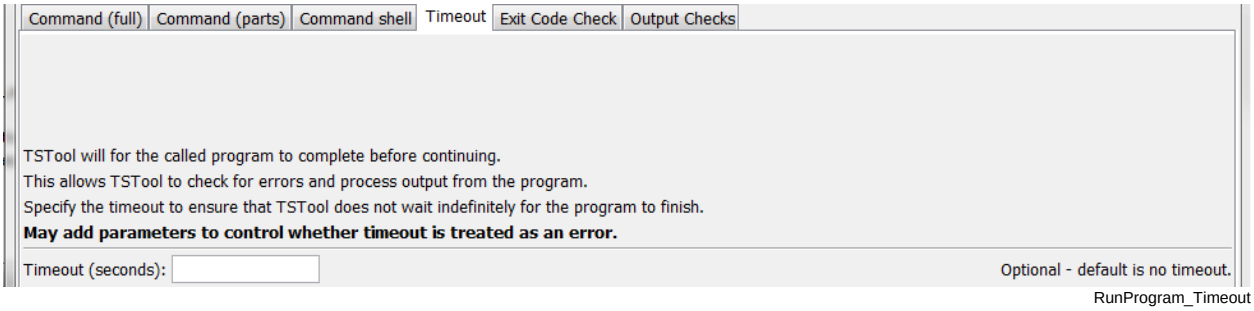
RunProgram_Parts

RunProgram() Command Editor when Specifying Command Line in Parts



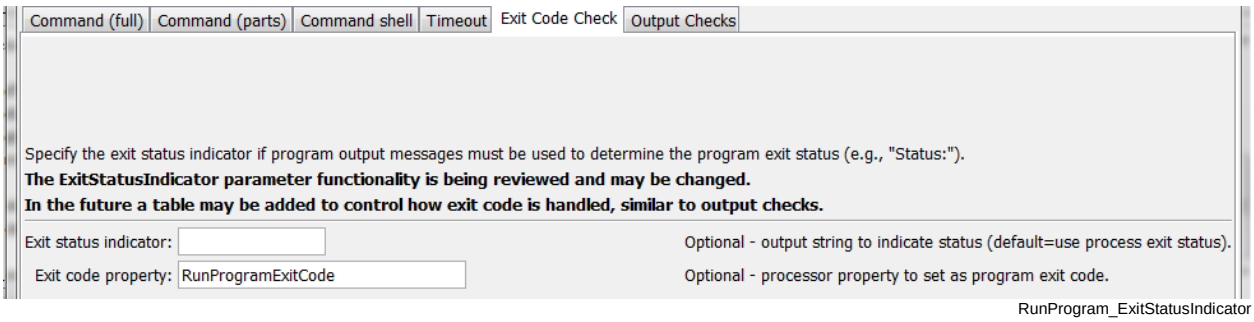
RunProgram_Shell

RunProgram() Command Editor showing Command Shell Parameters



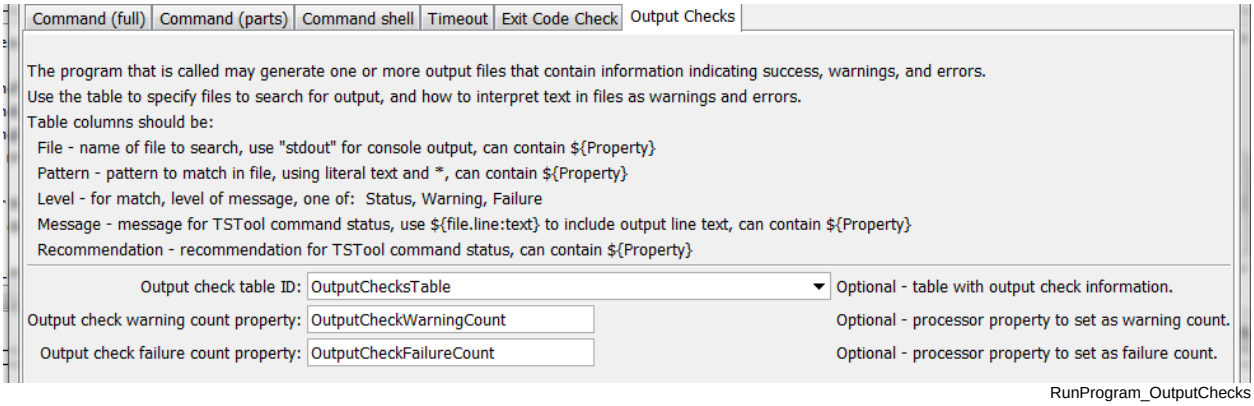
RunProgram_Timeout

RunProgram() Command Editor showing Timeout Parameters



RunProgram_ExitStatusIndicator

RunProgram() Command Editor showing Exit Code Parameters



RunProgram_OutputChecks

RunProgram() Command Editor showing Output Check Parameters

The command syntax is as follows:

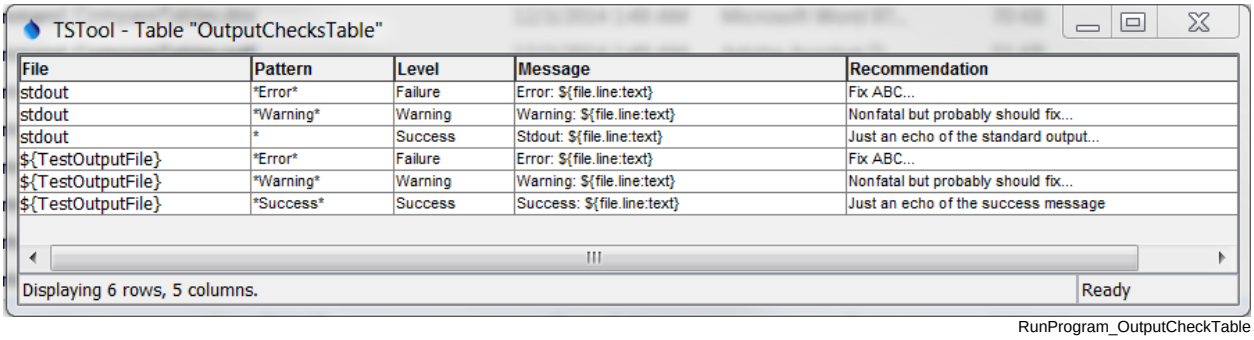
```
RunProgram(Parameter=Value…)
```

Command Parameters

Parameter	Description	Default
`CommandLine`	The full program command line, with arguments. If the program executable is found in the `PATH` environment variable, then only the program name needs to be specified. Otherwise, specify an absolute path to the program or run TSTool from a command shell the same directory. The `${WorkingDir}` property can be used in the command line to indicate the working directory (command file location) when specifying file names. Other `${Property}` names can also be used. For Windows, it may be necessary to place a `\"` at the start and end of the command line, if a full command line is specified.	Must be specified if the `Program` parameter is not specified. The `Program` parameter will be used if both are specified.
`Program`	The name of the program to run. Program arguments are specified using the `ProgramArg#` parameter(s). See the `CommandLine` parameter for more information about parameter formatting and locating the executable. Can specify with `${Property}`.	Must be specified if the `CommandLine` parameter is not specified.
`ProgramArg1, ProgramArg2, etc.`	Command like arguments used with `Program`. If necessary, use `${WorkingDir}` to specify the working directory to locate files. Can specify with `${Property}`.	No arguments will be used with `Program`.
`UseCommandShell`	If specified as `False`, the program will be run without using a command shell. A command shell	`True`, using `cmd.exe /C` on Windows and

Parameter	Description	Default
	is needed if the program is a script (batch file), a shell command, or uses >, |, etc.	`/bin/sh -c` on UNIX/Linux.
`CommandShell`	The command shell program to run for example on Windows: `cmd /c`. **Make sure that the shell is specified with an option to exit when the program completes (such as /c); otherwise, the process will hang.** Can specify with `${Property}`.	Determine automatically based on operating system.
`Timeout`	The timeout in seconds – if the program has not yet returned, the process will be ended. Zero indicates no timeout. **This behavior varies and is being enhanced**.	No timeout.
`ExitStatus Indicator`	This parameter may be phased out. Instead, use The `OutputCheckTableID` with a file of `stdout` and/or `ExitCodeProperty`. By default, the program exit status is determined from the process that is run. Normally 0 means success and non-zero indicates an error. However, the program may not exit with a non-zero exit status when an error occurs. If the program instead uses an output string like `STOP 3` to indicate the status, use this parameter to indicate the leading string, which is followed by the exit status (e.g., `STOP`).	Determine the exit status from the process exit value.
`ExitCode Property`	Name of the processor property to set as the exit code from the program being run. Can specify with `${Property}`.	Property is not set.
`OutputCheck TableID`	Table identifier for table containing output check patterns. Output file content can be scanned for patterns to detect success, warning, and errors. See the example file below for syntax of the table. Can specify with `${Property}`.	Output is not checked
`OutputCheck WarningCount Property`	Name of the processor property to set as the count of warning messages generated from the output table checks. Can specify with `${Property}`.	Property is not set.
`OutputCheck FailureCount Property`	Name of the processor property to set as the count of failure messages generated from the output table checks. Can specify with `${Property}`.	Property is not set.

The following figure illustrates the output check table specified by the `OutputCheckTableID` command.



File	Pattern	Level	Message	Recommendation
stdout	*Error*	Failure	Error: ${file.line:text}	Fix ABC...
stdout	*Warning*	Warning	Warning: ${file.line:text}	Nonfatal but probably should fix...
stdout	*	Success	Stdout: ${file.line:text}	Just an echo of the standard output...
${TestOutputFile}	*Error*	Failure	Error: ${file.line:text}	Fix ABC...
${TestOutputFile}	*Warning*	Warning	Warning: ${file.line:text}	Nonfatal but probably should fix...
${TestOutputFile}	*Success*	Success	Success: ${file.line:text}	Just an echo of the success message

RunProgram_OutputCheckTable

Example Output Check Table

The table columns are described below. The column names must be adhered to.

Output Check Table Column Descriptions

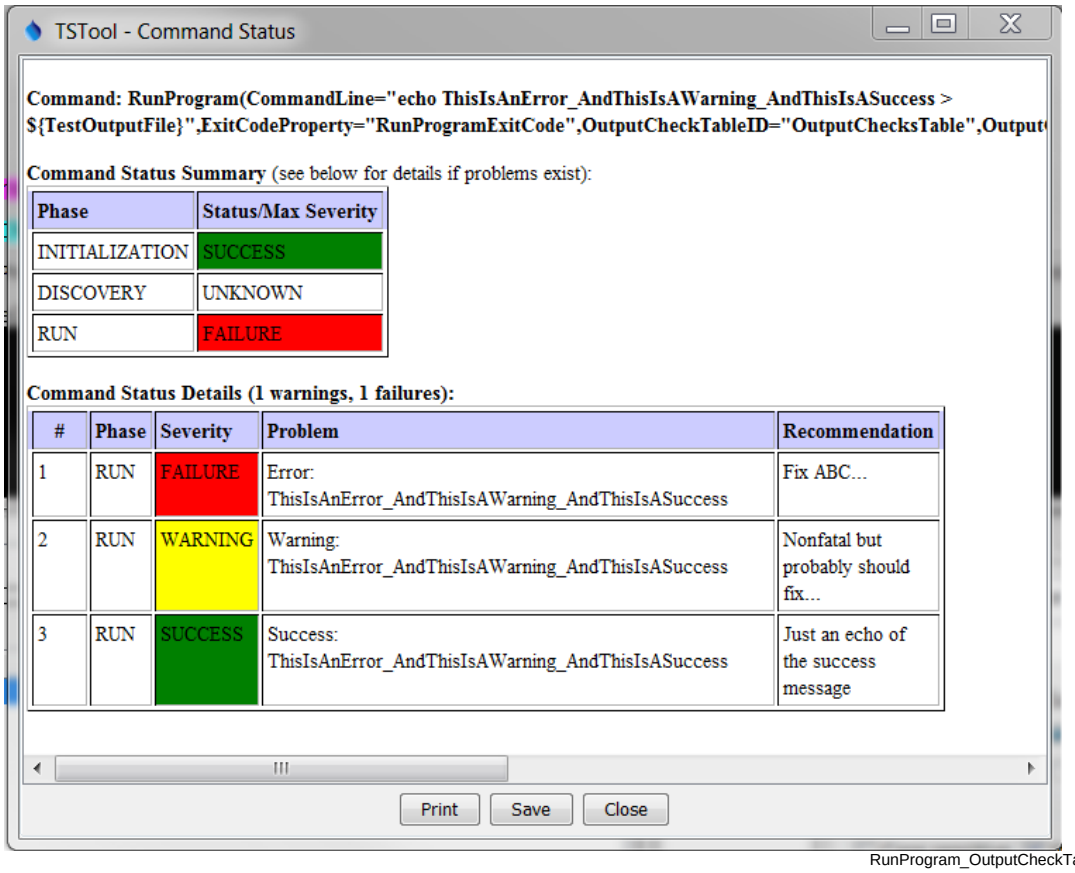
Parameter	Description
`File`	Name of the file to check or `stdout` to check standard output (console) output. Use `${WorkingDir}` to specify the location of the working directory (folder for command file). Can use the `${Property}` notation.
`Pattern`	The pattern to search for in the file. Can use * for wildcard. Searches are case-insensitive. Can use the `${Property}` notation.
`Level`	The message level to use in TSTool command status messages: • `Success` – use to echo messages to TSTool • `Warning` – will generate yellow warning indicators in TSTool • `Failure` – will generate red failure indicators in TSTool
`Message`	The message to include in TSTool command status messages, generally the cause of the issue. Can use the `${Property}` notation. The following special properties are recognized: • `${file.line:text}` – substitute entire file line • `${file.line:number}` – substitute output file line number • `${file:path}` – substitute full path for output file
`Recommendation`	A recommendation to fix the problem, as shown in TSTool command status messages. Can use the `${Property}` notation.

The following is an example that exercises the output check table (indentations represent line wrap).

```
# Test running an external program using a full command line with other defaults
# - use a command shell internally to run and determine the exit status from the
#   process exit value.
# - output to a file and check output using table of patterns
StartLog(LogFile="Results/Test_RunProgram_CommandLine_echo_OutputCheckTable.TSTool.log")
ReadTableFromDelimitedFile(TableID="OutputChecksTable",
  InputFile="Data\output-checks1.csv")
SetProperty(PropertyName="TestOutputFile",PropertyType=String,
  PropertyValue="${WorkingDir}/Results/
  Test_RunProgram_CommandLine_echo_OutputCheckTable_out.txt")
# Generate the output
RunProgram(CommandLine="echo ThisIsAnError_AndThisIsAWarning_AndThisIsASuccess >
```

```
  ${TestOutputFile}",ExitCodeProperty="RunProgramExitCode",
  OutputCheckTableID="OutputChecksTable",
  OutputCheckWarningCountProperty="OutputCheckWarningCount",
  OutputCheckFailureCountProperty="OutputCheckFailureCount")
If(Name="RunProgramWarningCheck",Condition="${OutputCheckWarningCount} > 0")
Message(Message="${RunProgramWarningCheck} warnings detected - might need to fix!",
  CommandStatus=WARNING)
EndIf(Name="RunProgramWarningCheck")
If(Name="RunProgramFailureCheck",Condition="${OutputCheckFailureCount} > 0")
Message(Message="${RunProgramFailureCheck} failures detected - definitely need to fix!",
  CommandStatus=FAILURE)
EndIf(Name="RunProgramFailureCheck")
WriteCheckFile(
  OutputFile="Results/Test_RunProgram_CommandLine_echo_OutputCheckTable_out.csv")
```

The status messages for the `RunProgram()` command from the above example are similar to the following.



RunProgram_OutputCheckTable_Status

Example Command Status Messages