Command Reference: RunCommands()

Run a command file

Version 3.08.02, 2010-01-07

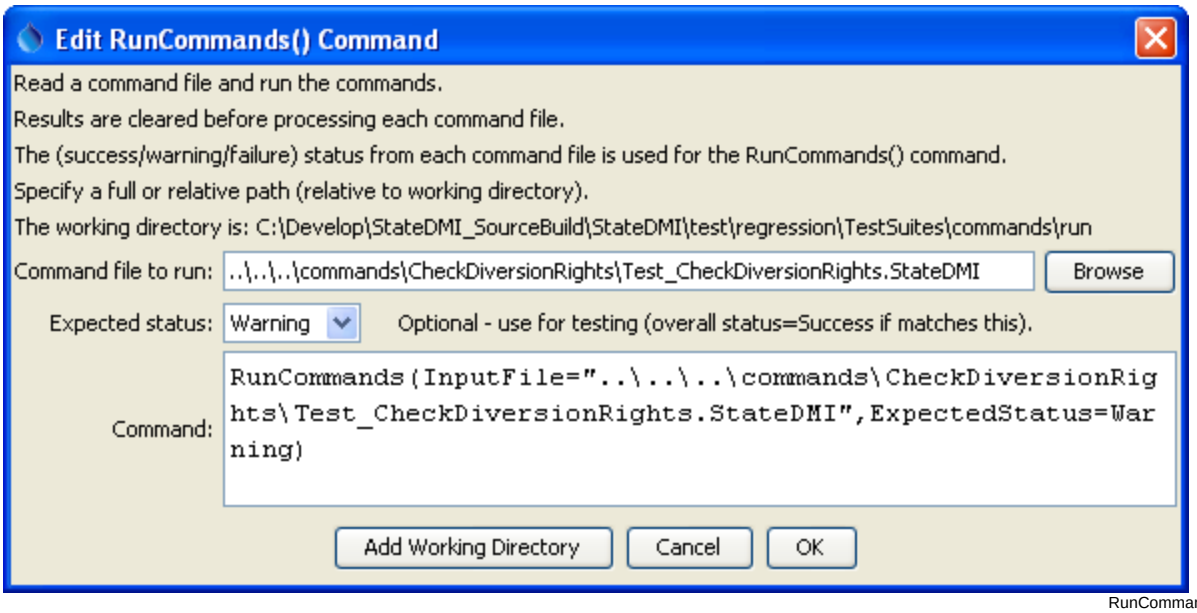
The `RunCommands()` command runs a command file. This command can be used to manage workflow where multiple commands files are run and is also useful for testing, where a test suite consists of running separate test case command files.

Command files that are run can themselves include `RunCommands()` commands. Each command file that is run has knowledge if its initial working directory and relative paths referenced in the command file are relative to this directory. This allows a master command file to reside in a different location than the individual command files that are being run. The current working directory is reset to that of the command file being run.

Currently the properties from the parent command file are NOT applied to the initial conditions when running the command file. Therefore, global properties like input and output period are reset to defaults before running the command file. A future enhancement may implement a property to indicate whether to inherit the properties. The output from the command is also not added to the parent processor. Again, a future enhancement may be to append output so that one final set of output is generated.

There is currently no special handling of log files; consequently, if the main command file opens a log file and then a command file is run that opens a new log file, the main log file will be closed. This behavior is being evaluated.

The following dialog is used to edit the command and illustrates the syntax for the command.



RunCommands

RunCommands() Command Editor

The command syntax is as follows:

```
RunCommands(Parameter=Value,…)
```

Command Parameters

Parameter	Description	Default
`InputFile`	The name of the command file to run, enclosed in double quotes if the file contains spaces or other special characters. A path relative to the master command file can be specified.	None – must be specified.
`ExpectedStatus`	Used for testing – indicates the expected status from the command, one of: • `Unknown` • `Success` • `Warning` • `Failure`	`Success`
`AppendResults`	Envisioned for implementation in the future. Indicate whether time series results from each command file should be appended to the overall time series results. This parameter currently always defaults to `False`, but support for `True` may be implemented in the future. Consequently, only the time series results from the last command file that is run will be displayed.	Currently always `False`

The following example illustrates how the `RunCommands()` command can be used to test software (or any implementation of commands that represent a standard process). First, individual command files are implemented to test specific functionality, which will result in warnings if a test fails:

```
# Test check diversion rights data where each checked value is in error
# The set command won't let invalid data be set from parameters so read bad data
# to trigger the check warnings.
# Compare the data csv to make sure the data are being produced as expected
# and the check file csv to make sure the checks are working.
# The expected status is Warning because the check will detect the missing values.
#@expectedStatus Warning
StartLog(LogFile="Results/Test_CheckDiversionRights.StateDMI.log")
RemoveFile(InputFile="Results\Test_CheckDiversionRights_out.csv",IfNotFound=Ignore)
RemoveFile(InputFile="Results\Test_CheckDiversionRights_out_check.csv",IfNotFound=Ignore)
RemoveFile(InputFile="Results\Test_CheckDiversionRights_out_check.html",IfNotFound=Ignore)
# Define a diversion station to trigger the check of stations
SetDiversionStation(ID="Diversion1",IfNotFound=Add)
SetDiversionRight(ID="Location1",IfNotFound=Add)
# Also read some bad data...
ReadDiversionRightsFromStateMod(InputFile="Data\simple.ddr")
# Uncomment the following command to regenerate the expected results.
#
WriteDiversionRightsToList(OutputFile="ExpectedResults/Test_CheckDiversionRights_out.csv")
WriteDiversionRightsToList(OutputFile="Results/Test_CheckDiversionRights_out.csv")
CompareFiles(InputFile1="ExpectedResults/Test_CheckDiversionRights_out.csv",
  InputFile2="Results/Test_CheckDiversionRights_out.csv",WarnIfDifferent=True)
#
# Check the data and create the check file.
CheckDiversionRights(ID="*")
# Uncomment the following command to regenerate the expected results.
# WriteCheckFile(OutputFile="ExpectedResults/Test_CheckDiversionRights_out_check.csv")
```

```
WriteCheckFile(OutputFile="Results/Test_CheckDiversionRights_out_check.csv")
WriteCheckFile(OutputFile="Results/Test_CheckDiversionRights_out_check.html")
CompareFiles(InputFile1="ExpectedResults/Test_CheckDiversionRights_out_check.csv",
  InputFile2="Results/Test_CheckDiversionRights_out_check.csv",WarnIfDifferent=True)
```

Next, use the `RunCommands()` command to run one or more tests:

```
StartRegressionTestResultsReport(
  OutputFile="RunRegressionTest_commands_general.StateDMI.out.txt")
…
RunCommands(
  InputFile="..\..\..\commands\CheckDiversionRights\Test_CheckDiversionRights.StateDMI",
  ExpectedStatus=Warning)…
```

Each of the above command files should produce expected results, without warnings. If any command file unexpectedly produces a warning, a warning will also be visible in StateDMI. The issue can then be evaluated to determine whether a software or configuration change is necessary.

This page is intentionally blank.