Test Case ID	*Test Description*	*Prerequisite*	*Test Steps*	*Verification Method*	*Input Data*	*Expected Results*
CTF-TC-01	Integer numbers comparison of with operators '<', '>', '<=', '>=', '==', '!='	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Start the cFS instance 2. Enable cFS telemetry output 3. Send **TO_RESET_CC** command to cFS instance 4. Send **TO_NOOP_CC** command to cFS instance 5. Compare **TO_HK_TLM_MID.usCmdCnt** with comparison operators '<' and '>' 6. Compare **TO_HK_TLM_MID.usCmdCnt** with comparison operators '<=' and '>=' 7. Compare **TO_HK_TLM_MID.usCmdCnt** with comparison operators '==' and '!=' 8. Shut down cFS	Test	1. None 2. None 3. None 4. None 5. Instruction arg as [{"variable": "usCmdCnt", "compare": "<","value": 3}, { "variable": "usCmdCnt", "compare": ">","value": 0 }] 6. Instruction arg as [{"variable": "usCmdCnt", "compare": ">=","value": 1}, { "variable": "usCmdCnt", "compare": "<=2","value": 1 }] 7. Instruction arg as [{"variable": "usCmdCnt", "compare": "==","value": 1}, { "variable": "usCmdCnt", "compare": "!=","value": 2 }] 8. None	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target and 'usCmdCnt' reset to 0. 3. The three CheckTlmValue instructions with comparison operators '<', '>', '<=', '>=','==', '!=' pass. 4. The running cFS instance is terminated and the xterm window closes.
CTF-TC-02	Comparison of string values	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Start the cFS instance 2. Enable cFS telemetry output 3. Compare **CFE_EVS_LONG_EVENT_MSG_MID.Payload.PacketID.AppName** with comparison operators 'streq' 4. Compare **CFE_EVS_LONG_EVENT_MSG_MID.Payload.PacketID.AppName**' with comparison operators 'strneq' 5. Compare **CFE_EVS_LONG_EVENT_MSG_MID.Payload.PacketID.AppName**' with comparison operators 'regex' 6. Shut down cFS	Test	1. None 2. None 3. Instruction arg as [{"variable": "Payload.PacketID.AppName", "compare": "streq", "value": "SCH" }] 4. Instruction arg as [{"variable": "Payload.PacketID.AppName", "compare": "strneq", "value": " SCHNO" }] 5. Instruction arg as [{"variable": "Payload.PacketID.AppName", "compare": "regex", "value": " S[CK]H" }] 6. None	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target. 3. The three CheckTlmValue instructions with comparison operators 'streq', "strneq', 'regex' pass. 4. The running cFS instance is terminated and the xterm window closes.
CTF-TC-03	Comparison with floating minus/plus tolerance values	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Start the cFS instance 2. Enable cFS telemetry output 3. Send '**DUMMY_IO_SET_CC**' command to cFS instance to set a fake telemetry output value. 4. Check **DUMMY_IO_OUT_DATA_MID.fValue** is within the plus tolerance. 5. Check **DUMMY_IO_OUT_DATA_MID.fValue** is within the minus tolerance. 6. Check **DUMMY_IO_OUT_DATA_MID.fValue** is within both the minus tolerance and plus tolerance. 7. Shut down cFS	Test	1. None 2. None 3. Instruction arg as { "fValue": 45.50} 4. Instruction arg as { "variable": "fValue", "compare": "==","value": 45.50, "tolerance_plus":0.01 } 5. Instruction arg as { "variable": "fValue", "compare": "==","value": 45.50, "tolerance_minus":0.01 } 6. Instruction arg as { "variable": "fValue", "compare": "==","value": 45.50, "tolerance_plus":0.01, "tolerance_minus":0.01 } 7. None	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target. 3. The three CheckTlmValue instructions pass, as the actual float value is within the expected value's tolerance ranges. 4. The running cFS instance is terminated and the xterm window closes.
CTF-TC-04	Build and start a cFS instance	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Set **build_cfs configuration** item to False in CTF configure file (.ini) 2. Remove cFS binary executable file, if it exists. 3. Set **json_results** configuration item to True in CTF config file (.ini) 4. Set log_level configuration itemto **DEBUG** or **INFO** or **ERROR** in CTF config file (.ini) 5. Add test instruction to build cFS, **BuildCfs** 6. Add test instruction to start the cFS instance, **StartCfs** 7. Enable cFS telemetry output 8. Send **TO_NOOP_CC** command to cFS instance 9. Shut down cFS after 30 seconds	Test and Inspection	1. Not applicable 2. Not applicable 3. Not applicable 4. Not applicable 5. None 6. None 7. None 8. None 9. None	1. cFS executable is re-built after executing BuildCfs instruction. 2. cFS target starts in a separate xterm session after executing StartCfs instruction . 3. CTF starts receiving telemetry from cFS target. 4. Telemetry and events data are logged in cfs_tlm_msgs.log and cfs_evs_msgs.log files in test results folder. 5. Test result summary are logged in results_summary.txt and results_summary.json files in test results folder. 6. Test execution data are logged in in test results folder. 'log_level' field in config file determines different levels of logging: with 'ERROR' level, it only show minimal output; with 'INFO' level, it shows info, warning, error l logs; with 'DEBUG' level, it shows all logs. 7. The running cFS instance is terminated and the xterm window closes.
CTF-TC-05	Run cFS app unit tests	1. The test environment is configured to run CTF. 2. cFS app unit tests are ready to build and run.	1. Init the SSH plugin 2. Send **SSH_RunRemoteCommand** to build unit test for CI app 3. Send **SSH_RunRemoteCommand** to execute CI unit test 4. Shut down CFS	Test	1. None 2. Instruction arg as { "command": "./build_unit_tests.sh" } 3. Instruction arg as { "command": "./run_unit_tests.sh" } 4. None	1. cFS unit tests are built and executed successfully for CI app.

Test Case ID	*Test Description*	*Prerequisite*	*Test Steps*	*Verification Method*	*Input Data*	*Expected Results*
CTF-TC-06	Execute multiple test scripts	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Pass multiple test script file names as arguments to CTF, '**./ctf ./scripts/example_tests/CTF-TC01.json ./scripts/example_tests/CTF-TC02.json**'	Test	1. ./scripts/example_tests/CTF-TC01.json ./scripts/example_tests/CTF-TC02.json	1. cFS target starts in a separate xterm window. 2. The first test script CTF-TC01.json passes. 3. CTF continues to execute the second test script CTF-TC02.json. 4. The second test script CTF-TC02.json passes. 5. The running cFS instance is terminated and the xterm window closes.
CTF-TC-07	Update non-continuous verification item with periodic telemetry during run-time	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	In test case one: 1. Start the cFS instance 2. Enable cFS telemetry output 3. Send **TO_RESET_CC** command to cFS instance 4. Send **TO_NOOP_CC** command to cFS instance 5. Verify **TO_HK_TLM_MID.usCmdCnt** to be 1 after 30 seconds In test case two: 6. Send **TO_NOOP_CC** command again to cFS instance. 7. Verify **TO_HK_TLM_MID.usCmdCnt** to be 2. 8. Shut down cFS	Test	1. None 2. None 3. None 4. None 5. Instruction arg as "args": { "variable": "usCmdCnt", "value": 1, "compare": "==" } 6. None 7. Instruction arg as "args": { "variable": "usCmdCnt", "value": 2, "compare": "==" } 8. None	1. The test script contains two test cases. 2. cFS target starts in a separate xterm window. 3. CTF starts receiving telemetry from cFS target and 'usCmdCnt' reset to 0 after step 3. 4. CTF executes the two test cases sequentially, 'usCmdCnt' is updated to 1 after step 4 in test case one. 5. 'usCmdCnt' is updated to 2 after step 6 in test case two. 6. Both test cases pass. 7. The running cFS instance is terminated and the xterm window closes.
CTF-TC-08	Verification with non-periodic telemetry	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Start the cFS instance 2. Enable cFS telemetry output 3. Send **CFE_ES_SHELL_CC** command to cFS instance 4. Verify **CFE_ES_SHELL_TLM_MID.Payload.ShellOutput** contains the sub-string No-op immediately 5. Verify the event "**TO - ENABLE_OUTPUT**" is not received from "**TO**" app for 10 seconds. 6. Shut down cFS	Test	1. None 2. None 3. Instruction arg as "Payload": { "CmdString":"echo No-op", "OutputFilename":"/ctf/test_output.txt" } 4. Instruction arg as {"compare":"regex", "variable":"Payload.ShellOutput", "value":"No-op*"} 5. Instruction arg as "data": {"app":"TO", "id":"3", "msg":"TO - ENABLE_OUTPUT ", "msg_args": "","target": ""}, "timeout": 10, 6. None	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target . 3. CheckTlmValue instruction passes. 4. The default verification timeout (5) is overridden to 10 in the instruction in step 5. 5. The running cFS instance is terminated and the xterm window closes.
CTF-TC-09	Continuous verification with periodic telemetry with updated verification items during run-time - passed case	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Start the cFS instance 2. Enable cFS telemetry output 3. Send **TO_RESET_CC** command to reset data item '**usCmdCnt**' to 0 4. Send **TO_NOOP_CC** command to cFS instance 5. Continuously verify **TO_HK_TLM_MID** data item '**usCmdCnt**' to be 1 6. Remove the continuous verification after 30 seconds 7. Shut down cFS after 30 seconds	Test and Inspection	1. None 2. None 3. None 4. None 5. Instruction arg as "args": { "variable": "usCmdCnt", "value": 1, "compare": "==" }, "verification_id": "usCmdCnt" 6 Instruction arg as "verification_id": "usCmdCnt" , "wait": 30 7. Instruction arg as "wait": 30	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target and 'usCmdCnt' reset to 0. 3. CheckTlmContinuous and RemoveCheckTlmContinuous instructions pass. 4. The test conductor verifies that CTF continuously captures and outputs the actual and expected values of verification item on terminal console and log files. 5. The running cFS instance is terminated and the xterm window closes.
CTF-TC-10	Continuous verification with periodic telemetry with updated telemetry value during run-time - failed case	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Start the cFS instance 2. Enable cFS telemetry output 3. Send **TO_RESET_CC** command to reset data item 'usCmdCnt' to 0 4. Send **TO_NOOP_CC** command to cFS instance 5. Continuously verify **TO_HK_TLM_MID** data item 'usCmdCnt' to be 1 6. Send **TO_NOOP_CC** command to cFS instance after 30 seconds. 7. Remove the continuous verification after 30 seconds 8. Shut down cFS.	Test and Inspection	1. None 2. None 3. None 4. None 5. Instruction arg as "args": { "variable": "usCmdCnt", "value": 1, "compare": "==" }, "verification_id": "usCmdCnt" 6. None 7. Instruction arg as "verification_id": "usCmdCnt" , "wait": 30 8. Instruction arg as "wait": 30	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target and 'usCmdCnt' reset to 0. 3. The test conductor verifies that CTF continuously captures and outputs the actual and expected values of verification item on terminal console and log files. 4. After step 6 (sending the second TO_NOOP_CC command), continuous verification generates warnings that the actual value of usCmdCnt telemetry does not match the expected value. 5. The running cFS instance is terminated and the xterm window closes.

Test Case ID	*Test Description*	*Prerequisite*	*Test Steps*	*Verification Method*	*Input Data*	*Expected Results*
CTF-TC-11	End test run upon the first instruction failure	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Set **end_test_on_fail** configuration item to **True** in CTF config file (.ini) 2. Start the cFS instance 3. Enable cFS telemetry output 4. Send **CI_NOOP_CC** command to cFS instance 5. Send **BAD_DATA** command to cFS instance 6. Send **TO_NOOP_CC** command to cFS instance 7. Shut down cFS	Test and Inspection	1. Not applicable 2. None 3. None 4. None 5. None 6. None 7. None	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target. 3. The instructions in step 2,3,4 pass. 4. The instruction in step 5 fails. 5. The running cFS instance is terminated and the xterm window closes. 6. The test conductor verifies that the remaining instructions are not executed and the test ends.
CTF-TC-12	Wait for user input to decide whether to continue the execution of a test case with "disable" attribute	1. The test environment is configured to run CTF 2. WaitForUserInput instruction is not added in 'ignored_instructions' field of CTF config file 3. A terminal console is ready for user input.	1. Execute **WaitForUserInput** instruction with attribute "disabled" set to true 2. Execute **WaitForUserInput** instruction with attribute "disabled" set to false 3. Enter user input from the terminal console	Test and Inspection	1. Instruction arg as "disabled" : true 2. Instruction arg as "disabled" :false 3. 'Y' or 'y'	1. The first WaitForUserInput instruction is skipped. 2. Test halts after executing the second WaitForUserInput instruction, and the console shows messages "Wait for user input: Please Enter 'Y' to continue" 3. Instruction will pass if entering 'Y' or 'y' from console. 4. Instruction will fail and the test aborts if entering anything other than 'Y' or 'y' from console.
CTF-TC-13	Execute conditional looping using user-defined variables	1. The test environment is configured to run CTF. 2. cFS is ready to build and run.	1. Start cFS the instance 2. Enable cFS telemetry output 3. Initialize a local variable **"my_var**" to 0 4. Initialize a local variable "**MaxLoopCnt**" to 3 5. Begin a loop and the looping condition is the local variable **"my_var**" is less than "**MaxLoopCnt**" During each loop, 6. Increase local variable "**my_var**" by 1 7. Send **TO_NOOP_CC** command to cFS instance After the conditional loop completes, 8. Check that the value of local variable "**my_var**" is 3	Test	1. None 2. None 3. Instruction arg as { "variable_name": "my_var", "operator": "=", "value": 0 } 4. Instruction arg as { "variable_name": "MaxLoopCnt", "operator": "=", "value": 3 } 5. Instruction arg as { "variable": "my_var", "compare": "<", "value": "$MaxLoopCnt$" } 6. Instruction arg as { "variable_name": "my_var", "operator": "+", "value": 1 } 7. None 8. Instruction arg as { "variable_name": "my_var", "operator": "==", "value": 3 }	1. cFS target starts in a separate xterm window. 2. CTF starts receiving telemetry from cFS target. 3. All instructions pass. 4. The running cFS instance is terminated and the xterm window closes.
CTF-TC-14	Interact with a single local cFS instance	1. The test environment must be configured to run CTF and cFS. 2. cFS is ready to build and run. 3. CTF is executed with the config file '**./vv_tests/configs/vv_lx1_config.ini**' and test script '**./vv_tests/scripts/vv_enable_output.json**'	1. Start cFS instance 2. Enable cFS output 3. Shut down cFS	Test	1. Not applicable 2. Not applicable 3. Not applicable	1. A new xterm window appears with cFS starting up. 2. cFS indicates receipt of TO_ENABLE_OUTPUT_CC. CTF starts receiving telemetry from cFS instance. 3. The xterm window is closed and the running cFS instance is terminated. The test passes.
CTF-TC-15	Execute a custom instruction defined in an external plugin	1. The test environment must be configured to run CTF and cFS. 2. cFS is ready to build and run. 3. CTF is executed with the config file '**./vv_tests/configs/vv_custom_plugin_config.ini**' and test script '**./vv_tests/scripts/vv_custom_plugin.json**'	1. Execute LogComment instruction	Test and Inspection	1. Instruction args { "msg": "This is a test comment." }	1. CTF console output logs the comment at ERROR level. 2. The test passes with exit code 0. 3. The test conductor inspects the output files to see that nothing was logged below ERROR level.
CTF-TC-16	Interacts with two remote Linux cFS instances simultaneously	1. The test environment must be configured to run CTF and cFS. 2. cFS is deployed on two remote targets, one Linux and one SP0. 3. The user running CTF must be able to SSH to the remote Linux host without a password prompt. 3. CTF is executed with the config file '**./vv_tests/configs/vv_lx1_lx2_remote_config.ini**' and test script '**./vv_tests/scripts/vv_enable_output.json**'	1. Start both cFS instances 2. Enable cFS telemetry output on both cFS instances 3. Shut down both cFS instances	Test	1. Not applicable 2. Not applicable 3. Not applicable	1. CTF console output indicates cFS starting up on both remote targets. 2. Both cFS instances indicate receipt of TO_ENABLE_OUTPUT_CC. CTF starts receiving telemetry from both cFS instances. 3. The cFS instances shuts down. The test passes with exit code 0.

Test Case ID	*Test Description*	*Prerequisite*	*Test Steps*	*Verification Method*	*Input Data*	*Expected Results*
CTF-TC-18	Interacts with two local cFS instances	1. The test environment must be configured to run CTF and cFS. 2. cFS is ready to build and run. 3. CTF is executed with the config file '**./vv_tests/configs/vv_lx1_lx2_config.ini**' and test script '**./vv_tests/scripts/vv_enable_output.json**'	1. Start both cFS instances 2. Enable cFS output on both instances 3. Shut down both cFS instances	Test	1. Not applicable 2. Not applicable 3. Not applicable	1. Two new xterm windows appears with cFS starting up in each. 2. Both cFS instances indicate receipt of TO_ENABLE_OUTPUT_CC. CTF starts receiving telemetry from both cFS instances. 3. The xterm windows are closed and both running cFS instances are killed. The test passes with exit code 0.
CTF-TC-19	Send an invalid CCSDS message	1. The test environment must be configured to run CTF and cFS. 2. cFS is ready to build and run. 3. CTF is executed with the config file '**./vv_tests/configs/vv_lx1_config.ini**' and test script '**./vv_tests/scripts/vv_invalid_command.json**'	1. Start the cFS instance 2. Enable cFS telemetry output 3. Send TO_NOOP_CC command with an invalid payload length of 10 4. Shut down cFS	Test	1. Not applicable 2. Not applicable 3. Instruction args as { "mid": "TO_CMD_MID", "cc": "TO_NOOP_CC", "payload_length": 10 } 4. Not applicable	1. A new xterm window appears with cFS starting up. 2. cFS indicates receipt of TO_ENABLE_OUTPUT_CC. CTF starts receiving telemetry from cFS instance. 3. cFS indicates receipt of TO_NOOP_CC with an invalid length. 4. The xterm window is closed and the running cFS instance is killed. The test passes with exit code 0.
CTF-TC-20	Create/modify test scripts via CTF graphical editor	1. The test environment is configured to run CTF. 2. The test environment is configured to run the CTF graphical editor.	1. Start the CTF editor 2. Open the CTF workspace 3. Create a new empty script 4. Add new test 5. Add new instruction to test script 6. Modify instruction's arguments 7. Save modified test script	Inspection	1. From 'File' menu, click 'Open Workspace' submenu. From the pop-up window, navigate the file system to open the workspace config file. An example config file is 'editor_workspace.json' file in the folder of sample_cfs_workspace/ctf_tests 2. From 'File' menu, click 'New Test Script' submenu to create a new empty script. 3. On Editor Pane of UI, click 'ADD TEST' button to add a new test. 4. From the Command Palette, click 'CFS PLUGIN' tab to expand it. Then drag and drop the 'StartCfs' instruction to the expanded area of new script. 5. Click the added 'StartCfs' instruction button, from the popover window, edit its arguments 6. From 'File' menu, click 'Save Test Script' submenu to save it.	The test conductor verifies the followings: 1. Before opening workspace config file, the File Pane of UI is empty; After opening config file, File Pane is updated to a tree structure file explorer. 2. After creating new script, the Editor Pane shows an empty script. 3. After clicking 'ADD TEST' button, a 'Untitled Test' empty test is created. 4. After expanding 'CFS PLUGIN' tab on Command Palette, 15 instructions under CFS plugins are displayed and ready to be dropped into the new test on Editor Pane. After 'StartCfs' instruction is added, the test has one instruction. 5. Clicking 'StartCfs' instruction button, a popover window appears to modify its arguments. 6. After clicking 'Save Test Script' menu, a pop-up window appears, tester can enter the file name and choose location to save the script file. 7. Test conductor opens the saved script file and verifies the content of the test script is modified as expected.
CTF-TC-21	Auto-suggest MID and CC input via CTF graphical editor	1. The test environment is configured to run CTF. 2. The test environment is configured to run the CTF graphical editor.	1. Start the CTF editor 2. Open the CTF workspace 3. Select and open test script 4. Select 'SendCfsCommand' instruction 5. Modify 'mid' input. 6. Modify 'cc' input.	Inspection	1. From 'File' menu, click 'Open Workspace' submenu. From the pop-up window, navigate the file system to open the workspace config file. An example config file is 'editor_workspace.json' file in the folder of sample_cfs_workspace/ctf_tests 2. From File Pane of UI, select the test script (For example TC-21) and click it. 3. On Editor Pane of UI, click the test script to expand its instructions, then click 'SendCfsCommand' instruction. 4. On the instruction's popover window, click the 'mid' input box, select the mid code, such as 'TO_CMD_MID'. 5. On the instruction's popover window, click the 'cc' input box, select the cc code, such as 'TO_NOOP_CC'.	1. Before opening workspace config file, the File Pane of UI is empty; After opening config file, File Pane is updated to a tree structure file explorer. 2. After selecting (clicking) the test script on File Pane, the Editor Pane of UI loads the test details - Owner, Description, Instructions, etc. 3. After clicking the test script on Editor Pane, it expands to display the details of its instructions. Click 'SendCfsCommand' button, a popover window appears to modify its arguments. 4. The 'mid' input box auto-suggests available mid numbers. As more characters are entered, the suggested mid numbers narrows. 5. Similar to mid, the 'cc' input box auto-suggests available cc codes based on the entered mid. As more characters are entered, the suggested cc codes narrows.
CTF-TC-22	Load workspace configuration for the graphical editor	1. The test environment is configured to run CTF. 2. The test environment is configured to run the CTF graphical editor.	1. Start the CTF editor 2. Open the CTF workspace	Inspection	1. From 'File' menu, click 'Open Workspace' submenu. From the pop-up window, navigate the file system to load the workspace config file. An example config file is 'editor_workspace.json' file in the folder of sample_cfs_workspace/ctf_tests	1. Before loading workspace config file, the File Pane of UI is empty, the Editor Pane has No Data, the Command Palette and the Function Palette are None. 2. After loading config file, File Pane is updated to a tree structure directory. The root is the setting of "scriptDir" in the config file. The Command Palette is also updated to add the plugins' instructions, which comes from 'pluginDir' setting in the config file.

Test Case ID	*Test Description*	*Prerequisite*	*Test Steps*	*Verification Method*	*Input Data*	*Expected Results*
CTF-TC-23	Start CTF test execution via graphical editor	1. The test environment is configured to run CTF. 2. The test environment is configured to run the CTF graphical editor. 3. cFS is ready to build and run.	1. Start the CTF editor 2. Open the CTF workspace 3. Select and open the test script 4. Run the the test script	Inspection	1. From 'File' menu, click 'Open Workspace' submenu. From the pop-up window, navigate the file system to open the workspace config file. An example config file is 'editor_workspace.json' file in the folder of sample_cfs_workspace/ctf_tests 2. From File Pane of UI, navigate and select the test script to run. 3. Right click the script, from the menu, click 'Run (Default Config)'.	1. Before opening workspace config file, the File Pane of UI is empty; After opening config file, File Pane is updated to a tree structure file explorer. 2. After selecting (clicking) the test script, the Editor Pane of UI loads the test details - Owner, Description, Instructions, etc. 3. After right clicking the test script, a menu pops up with 4 items: "Run (Default Config)" "Run (Custom Config)" "Rename" "Delete" 4. After clicking 'Run (Default Config)', a pop-up window 'Run status' appears. And cFS target starts on a separate xterm window. 5. As the test executes, the instructions' icons on the window change status (pass or fail) in real time. 6. After test completes, xterm window disappear. And the button at the bottom of the window changes the caption from 'Running' to 'Done'.
CTF-TC-24	Stop test execution via CTF graphical editor	1. The test environment is configured to run CTF. 2. The test environment is configured to run the CTF graphical editor. 3. cFS is ready to build and run.	1. Start the CTF editor 2. Open the CTF workspace 3. Select and open the test script 4. Run the test script 5. Cancel the test before its completion	Inspection	1. From 'File' menu, click 'Open Workspace' submenu. From the pop-up window, navigate the file system to open the workspace config file. An example config file is 'editor_workspace.json' file in the folder of sample_cfs_workspace/ctf_tests 2. From File Pane of UI, navigate and select the test script to run. 3. Right click the script, from the menu, click 'Run (Default Config)', a pop-up window 'Run status' appears. 4. Before the test completes, click the 'Cancel' button at the bottom of 'Run status' window.	1. Before opening workspace config file, the File Pane of UI is empty; After opening config file, File Pane is updated to a tree structure file explorer. 2. After selecting (clicking) the test script, the Editor Pane of UI loads the test details - Owner, Description, Instructions, etc. 3. After right clicking the test script, a menu pops up with 4 items: "Run (Default Config)" "Run (Custom Config)" "Rename" "Delete" 4. After clicking 'Run (Default Config)', a pop-up window 'Run status' appears. And cFS target starts on a separate xterm window. 5. After clicking 'Cancel' button (before the test completes), the test stops, the pop-up window and xterm window disappear.
CTF-TC-25	Display test instruction status immediately after execution via CTF graphical editor	1. The test environment is configured to run CTF. 2. The test environment is configured to run the CTF graphical editor. 3. cFS is ready to build and run.	1. Start the CTF editor 2. Open the CTF workspace 3. Select and open the test script 4. Run the test script 5. Check the execution status of each test instruction	Inspection	1. From 'File' menu, click 'Open Workspace' submenu. From the pop-up window, navigate the file system to open the workspace config file. An example config file is 'editor_workspace.json' file in the folder of sample_cfs_workspace/ctf_tests 2. From File Pane of UI, navigate and select the test script to run. 3. Right click the script, from the menu, click 'Run (Default Config)' 4. Move mouse to instruction status button on 'Run status' pop-up window.	1. Before opening workspace config file, the File Pane of UI is empty; After opening config file, File Pane is updated to a tree structure file explorer. 2. After selecting (clicking) the test script, the Editor Pane of UI loads the test details - Owner, Description, Instructions, etc. 3. After right clicking the test script, a menu pops up with 4 items: "Run (Default Config)" "Run (Custom Config)" "Rename" "Delete" 4. After clicking 'Run (Default Config)', a pop-up window 'Run status' appears. In the window, test instructions are organized in a tree structure. The status button of each instruction has 3 states (icons): grey '...' icon means the test is not executed green 'check' icon means the test passes red 'error' icon means the test fails 5. cFS target starts on a separate xterm window. 6. As test goes, the icons will change to reflect instructions' test status in real time. Placing the mouse on the button, a popover message will appear for more details.

Test Case ID	*Test Description*	*Prerequisite*	*Test Steps*	*Verification Method*	*Input Data*	*Expected Results*
CTF-TC-26	Execute conditional branching test instructions	1. The test environment is configured to run CTF.	In test case one: 1. Initialize a local variable "**my_var**" to 1 2. Check whether the variable "**my_var"** is 1. 3. If the variable "**my_var**" is 1, set the variable "**my_var**" to 10. 4. Else, set the variable "**my_var**" to -10. 5. Verify the variable "**my_var**" to be 10. In test case two: 6. Check whether the variable "**my_var**" is 1. 7. If the variable "**my_var**" is 1, set the variable "**my_var**" to 10. 8. Else, set the variable "**my_var**" to -10. 9. Verify the variable "**my_var**" to be -10.	Test	1. Instruction arg as { "variable_name": "my_var", "operator": "=", "value": 1 } 2. Instruction arg as { "variable_name": "my_var", "operator": "==", "value": 1 } 3. Instruction arg as { "variable_name": "my_var", "operator": "=", "value": 10 } 4. Instruction arg as { "variable_name": "my_var", "operator": "=", "value": -10 } 5. Instruction arg as { "variable_name": "my_var", "operator": "==", "value": 10 } 6. Instruction arg as { "variable_name": "my_var", "operator": "==", "value": 1 } 7. Instruction arg as { "variable_name": "my_var", "operator": "=", "value": 10 } 8. Instruction arg as { "variable_name": "my_var", "operator": "=", "value": -10 } 9. Instruction arg as { "variable_name": "my_var", "operator": "==", "value": -10 }	1. In test case one, instruction in Step 4 is skipped, as the conditional branch condition evaluation is true. 2. In test case two, instruction in Step 7 is skipped, as the conditional branch condition evaluation is false. 3. The two test cases pass.