WHEN YOU NEED TO BE SURE

Monitoring Test Cell State (Variables) with state_mon

Version 5

May 31, 2022

Developed by SGS North America, Inc.

Version History

Version	Date	Revision Description
1	1/25/2016	Initial publication
2	8/23/2018	Format with SGS brand
3	4/3/2020	Retrofit to new template
4	12/2/2021	Added hypertext linked cross-reference to cyflex.com usage help for `state_mon` in *Section 1* Overview on page 1
5	5/31/2022	Updated hypertext linked cross-reference to cyflex.com usage help for `state_mon` in *Section 1 Overview* on page 1

Document Conventions

This document uses the following typographic and syntax conventions.

- Commands, command options, file names or any user-entered input appear in Courier type. Variables appear in Courier italic type.

  Example: Select the `cmdapp-`*`relVersion-buildVersion`*`.zip` file….

- User interface elements, such as field names, button names, menus, menu commands, and items in clickable dropdown lists, appear in Arial bold type.

  Example: **Type**: Click **Select Type** to display drop-down menu options.

- Cross-references are designated in Arial italics.

  Example: Refer to *Figure 1*…

- Click intra-document cross-references and page references to display the stated destination.

  Example: Refer to *Section 1* Overview on page 1.

  The clickable cross-references in the preceding example are *1*, *Overview*, and on page 1.

CyFlex Documentation

CyFlex documentation is available at https://cyflex.com/. View **Help & Docs** topics or use the **Search** facility to find topics of interest.

Table of Contents

1 Overview

Occasionally there is a need to periodically check a group of variables and allow different action to be performed depending on the combined value of the variables. For example:

- If all the following are TRUE at any time:
  - No flames are detected in the test cell.
  - The test cell doors are closed.
  - The ignition key switch is turned on.
  - The combustion air temperature is less than the allowed maximum.
- Then, the engine may be started.
- Otherwise, the engine may not be started.

To accomplish the preceding, using basic CyFlex Test Manager (`gp_test`) capabilities could result in a test procedure that is complicated. This is particularly true if many variables need to be checked. To better address this, use the `gp_test` support process `state_mon`. This process is called from a `gp_test` test mode and checks the values of a group of variables. The checked variables are specified in a specification file(s) read by `state_mon`. When the variables are in the appropriate state, `state_mon` signals `gp_test` to proceed to the next appropriate test mode.

Refer to *Introduction to state_mon* and cyflex.com usage help for `state_mon` for additional reference.

2 state_mon Modes of Operation

When `state_mon` is called in a `gp_test` mode, the calling arguments define how the state variables specified in a spec file, should be watched. There are three ways that the state variables may be watched:

1. `MONITOR` mode to verify that all variables remain in the specified states
2. `IMMEDIATE` mode to determine the current state of the variables
3. `VERIFY` mode to verify that all variables have reached the specified states

Each specified variable has an action, (e.g. equality, `EQ`) associated with it. In addition, each action may include one of three possible extensions (e.g. critical, `_C`). As a result, the state variables fall into one of two groups:

1. Variables with action extensions specified
2. Variables without action extensions

Refer to *Section 2.1.1 Specifying State Variables* on page 3.

When `state_mon` is called, the current value of each variable is checked. If any of the variables with an action extension specified fail, then `gp_test` is immediately signaled to proceed to one of the three return modes associated with the extension. This occurs regardless of the `state_mon` mode of operation except for the case when variables that have an action specified without an extension.

2.1 Mode of Operation for Actions with no Extensions

When `state_mon` is called with the `MONITOR` mode specified, it is assumed that all the variables have the correct value when the mode is executed. The variables are periodically checked at the rate specified in the specification file. If any of the variables fail the state check, the `gp_test` procedure advances to the next appropriate mode. Otherwise, the test procedure will remain in the mode until a variable fails or a 'timeout' occurs Refer to *Section 3 gp_test Keywords* on page 4.

When called with the `IMMEDIATE` mode specified, each variable is checked once. After they have all been checked, the procedure advances to the next appropriate mode. As a result, the next mode may be one of five possible return modes associated with the state variables. (four different failure returns and a success return.

When called in the `VERIFY` mode, it is assumed that some variables that have associated actions without extensions do not have the specified value. While in this `gp_test` mode the value of all variables are periodically checked. The test remains in the mode until all variables reach their specified state, then the test advances to the 'success' mode.

Also, an optional timeout specification will cause `state_mon` to advance to the specified timeout mode. This will occur regardless of the values of the state variables. The timeout value is the maximum time to check the variables.

2.1.1 Specifying State Variables

To perform the function associated with each of the three possible modes of operations, `state_mon` must know the following:

- The variables to watch
- The action associated with each variable
- The desired state of each variable
- The periodic rate that the variables should be checked

The desired value of a specified state variable may be different depending on which step is being executed in the testing process. As a result, there can be groups of desired values for each specified variable. Therefore, there needs to be a mechanism available to `state_mon` that identifies which group of specified states to use when checking the variables. This is accomplished by associating a string or 'index' with each group of states. This information, as well as the above four items, is contained in a specification file(s) that is read by `state_mon`.

When `state_mon` is called from a `gp_test` mode, it receives information that indicates:

- The specification file(s) that should be read
- Which group of state values should be watched
- The desired mode for `state_mon`

Once the specification file has been read, the checking of the variables begins. Depending on the mode of `state_mon`, after each check of the variables `gp_test` may be signaled to go to the next appropriate mode. For example, assume the following:

- All the state variables that have an action specified with an extension equals the specified value .i.e. the combined state of the variables is TRUE. Refer to *Appendix A. state_mon actions for State Variables* on page 9.
- Some of the state variables that have an action specified without an extension are not equal to the specified state; .i.e. the combined state of these variables is FALSE.

If the `state_mon` mode is `VERIFY`, then continue to check the state variables value

If the `state_mon` mode is `MONITOR`, then `gp_test` is signaled to go to the failure mode

If the `state_mon` mode is `IMMEDIATE`, then `gp_test` is signaled to go to the failure mode.

Note:

`gp_test` is always signaled after the first check of the state variables value for this mode.

3 gp_test Keywords

As previously mentioned, `state_mon` may be in one of three modes, `VERIFY`, `MONITOR`, or `IMMEDIATE`. `gp_test` will be signaled to proceed to the next appropriate mode when the state variables have reached the appropriate value. The *appropriate* value depends on the mode of `state_mon`.

The configuration information needed by `state_mon` is provided via `gp_test` keywords that are specified in a particular test mode. Refer to the *Test Manager User Guide* for details on developing a complete test procedure. The `state_mon` keywords are:

- `@STATE_MON_ACTIONS`
- `@STATE_MON_SPEC_FILES`
- `@STATE_MON_EXCEPTIONS`

Refer to the following:

- *Appendix D. gp_test Keywords for state_mon* on page 14 for an example `gp_test` procedure specification.
- Appendix *E. gp_test Trace Information Generated by state_mon* on page 15 for the information that is written to the TRACE file when the test procedure is in a `state_mon` mode

3.1 @*STATE_MON*_ACTIONS

`<successful return path> <failure return path> <read mode> <state_mon mode>`

Where:

- `<successful return path>` is the next mode to be executed when *state_mon* returns normally. This may be `MODE_TERMINATE`, `NONE`, another number, or another `gp_test` procedure.
- `<failure return path>` is the next mode to be executed when *state_mon* returns with a failed condition. This may be `MODE_TERMINATE`, `NONE`, another number, or another `gp_test` procedure.
- `<read mode>` informs `state_mon` how to read the specification file(s). Valid entries for this field are `READ` or `READ_ONCE`. For `READ`, the spec file is read every time the test mode is executed. However, for `READ_ONCE`, the spec file is read only once.
- `<state_mon mode>` is the desired mode for *state_mon* when checking the variables. Valid entries are `VERIFY`, `MONITOR`, and `IMMEDIATE`.

3.2 @*STATE_MON*_SPEC_FILES

```
<spec file pathname>                    <state index>
```

Where:

- <spec file pathname> is the full pathname of the *state_mon* specifications. There can be a maximum of sixteen specification files.
- <state_index> is the label of the variable that will contain the index value that indicates which group of state values should be read from the file. This can also be thought of as which *column* should be processed. (VERTICAL_LABELS file format).

3.3 @*STATE_MON*_EXCEPTIONS

There are also exceptions that will result in the gp_test procedure advancing to the specified exception mode. The exception mode to be taken is specified by the extension to the specified action for each variable. These modes are specified via the @STATE_MON_EXCEPTION gp_test keyword. The exceptions are:

- The specified length of time to check the variables has expired.
- A variable with a critical action extension (_C) has failed its check.
- A variable with a warning action extension (_W) has failed its check.
- A variable with a state_change' action extension (_S) has failed its check.
- An error occurred when reading the specification file.

@*STATE_MON*_EXCEPTIONS

```
<timeout> <timeout> <state change> <critical failure> <warning failure> <read error>
          <path>        <path>           <path>           <path>           <path>
```

Where:

- <timeout> is the maximum length of time to allow the variables to be checked. Valid entries are a value, a CyFlex label, or a computed expression. For example, 23[sec]. This entry, although valid for all modes, is only relevant for the VERIFY and MONITOR modes.
- <timeout path> is the path that should be taken when a timeout occurs. This may be MODE_TERMINATE, NONE, another number, or another gp_test procedure.
- <state change path> is the path that should be taken when a variable with an action extension of _S fails its check.
- <critical failure path> is the path that should be taken when a variable with an action extension of _C fails its check.
- <warning failure path> is the path that should be taken when a variable with an action extension of _W fails its check.
- <read error path> is the path that should be taken when an error is detected reading the specification file.

3.4 state_mon output to gp_test

The information that *state_mon* returns to `gp_test` contains the state of seven different conditions,

1. Critical variable failure
2. Warning variable failure
3. State change failure
4. Failure return
5. Success return
6. The length of time to check the state variables has expired
7. An error occurred reading the specification file

3.5 Termination Paths

Since more than one condition may exist that imply different termination paths, there are rules of precedence for the termination paths. Assuming that there are paths specified for all exceptions, the rules of precedence are:

If an error occurred reading the specification file, then take the `read_error` termination path

If a timeout exception occurred, then take the `timeout` termination path.

If a `_S` variable failed, then take the `state_change` termination path.

If a `_C` variable failed then, take the `critical` termination path.

If a `_W` variable failed, then take the `warning` termination path.

If a `failure` return occurred, then take the `failure` return.

If a `success` return occurred, then take the `success` return.

4 Specification File Content

The specification file, a flat ASCII file, provides the following basic information to *state_mon*:

- A list of variables that should be checked
- How often the variables should be checked
- A list of values that should be associated with each variable
- A list of indices that should be assigned to each value specified for the variable

The specification file may have one of two fundamental formats:

- The variables to be watched may be listed on a single line in HORIZONTAL_LABELS format. This format is typically used when there are a limited number of variables to watch. Refer to *Appendix B. Horizontal Labels state_mon Specifications* on page 12.
- The variables to be watched may be listed on multiple lines in VERTICAL_LABELS format. This format is typically used when there are many variables to be watched. Refer to *Appendix C. Vertical Labels state_mon Specifications* on page 13.

For each variable, an action code and, optionally, an extension may be specified. The action code indicates the action that should be performed when it is checked. The extension indicates how a state failure should be interpreted. For example, the action may indicate that an equality (EQ) test should be performed and that the extension indicates that the test is 'critical'. This would result in the complete action being EQ_C. Refer to *Appendix A. state_mon actions for State Variables* on page 9.

For the file format HORIZONTAL_LABELS, specify each variable and action on a single line in the text file. Specify the desired state value for each of the specified variables on lines following the specified variables. Preceding the values is the string or state index that will be used to identify each group of values. Additional lines may be added for each additional state index and group of state values. There must be an entry specified for each specified variable. The first specified state value would be associated with the first specified variable; the second state value would be associated with the second variable

If the file format is VERTICAL_LABELS, the specified state indices and variables are transposed. Specify all the state indices that will be used to identify a group of state values on a single line. Specify the state values for each of the state indices are on a single line preceded by the variable specification. Specify one state value for each state index. The association of the state values is also transposed.

- For the VERTICAL_LABELS format, each state value on a given line is associated with each specified state index.
- If the file format is HORIZONTAL_LABELS, each state value, on a given line, is associated with each specified variable.

Each state value may have additional information associated with it. For example, if the state action for a variable is a limit violation check, then the value may have a time associated with it. The time is the specified length of time the limit must be violated before the limit will actually be violated. Refer to *Appendix A. state_mon actions for State Variables* on page 9 for the details on specifying state values.

Consider the excerpt below from a spec file that has the VERTICAL_LABELS format. The variable being monitored is oilrfl_t and the monitor action is a low limit that has an extension of warning. There are three possible 'values' for the lower limit. In two cases, the value is the value of the variable cmp_in_t. For the third case, the value of oilrfl_t will not be monitored, since the specified value of DC (Don't Care) was given. When the state index variable, as specified under the gp_test @*STATE_MON*_SPEC_FILES keyword, has a value of 44, then oilrfl_t will be monitored. If its value is less than the value of cmp_in_t, then the state of oilrfl_t will be failed since the lower limit was violated. What happens in the gp_test mode that called state_*mon* depends on the action code that was specified under the @STATE_MON_ACTIONS keyword. If the action code was VERIFY, then the variable will continue to be watched. However, if the action code was MONITOR, gp_test will be signaled to exit the current mode. If a path was specified for a 'failure' as part of the @STATE_MON_ACTIONS keyword, that path is taken. Otherwise, the 'default next mode' is taken.

```
# state
# index              index value         index value     index value

@STATE_INDICES
                           44                  twelve           5


@STATE_VALUES_TABLE

# variable:action        value               value              value

oilrfl_t:LO            cmp_in_t            cmp_in_t:4[sec]       DC
```

Appendices

Appendix A. state_mon actions for State Variables

The variable to be watched is specified in the *state_mon* spec file under one of two keywords.

- If the spec file format is `VERTICAL_LABELS`, it is specified as the first field of each entry under the `@STATE_VALUES_TABLE` keyword.
- If the file format is `HORIZONTAL_LABELS`, it is specified under the `@STATE_VARIABLES` keyword.

In either case, the format of the state variable entry is as follows:

`variable_lbl:action`

Where:

- `<variable_lbl>` is the label of the Cyflex variable that is to be watched.
- `<action>` is the type of action that should be performed when watching the variable.

The `action` field falls into the following classes and it must be one of the strings (case-insensitive) listed for each class.

- Limits
  - `LO_C` < lower limit – critical
  - `LO_W` < lower limit – warning
  - `LO_S` < lower limit – state change
  - `LO` < lower limit
  - `UP_C` < upper limit – critical
  - `UP_W` < upper limit - warning
  - `UP_S` < upper limit – state_change
  - `UP` < upper limit
- Equality
  - `NE_C` < not equal – critical
  - `NE_W` < not equal – warning
  - `NE_S` < not equal – state change
  - `NE` < not equal
  - `EQ_C` < equal – critical
  - `EQ_W` < equal - warning
  - `EQ_S` < equal - state change
  - `EQ` < equal
- Statistical Deviation
  - `DV_C` < deviation – critical
  - `DV_W` < deviation – warning
  - `DV_S` < deviation – state change
  - `DV` < deviation

- Statistical Variance
  - VR_C < variance – critical
  - VR_W < variance – warning
  - VR_S < variance – state change
  - VR < variance
  - CV_C < coef of var – critical
  - CV_W < coef of var – warning
  - CV_S < coef of var – state change
  - CV < coef of var
- Time Delay; for this action, the variable is a dash (-)
  - TD_C < time delay – critical
  - TD_W < time delay - warning

All values for a state variable that are specified under the @STATE_VALUES_TABLE keyword may be computed expressions. The formats of the value entry for the above classes are:

- Limits

  `state_value:window_duration`

  where:
  - `state_value` is the value of the limit
  - `window_duration` is the time the variable must be violated before the limit is actually violated.
- Equality

  `state_value`

  where:
  - `state_value` is the value for the equality test
- Statistical Deviation

  `state_value:tolerance:window_duration`

  where:
  - `state_value` is the base value used to determine the minimum and maximum values.
  - `tolerance` is the value that is added or subtracted to the `state_value` to determine the minimum and maximum allowed values.
  - `window_duration` is the time window width for the statistical variable.
- Statistical Variance

  `state_value:window_duration`

  where:
  - `state_value` is the base value used to determine the minimum and maximum values.
  - `window_duration` is the time window width for the statistical variable.

- Time Delay

  `state_value`

  where:

  - `state_value` is the value of the time delay

Appendix B. Horizontal Labels state_mon Specifications

```
@FILE_FORMAT
   HORIZONTAL_LABELS

      # the following defines the rate at which the variables are
      # monitored

@PROCESS_INTERVAL
   1000

@STATE_VARIABLES
       Engine_Run:Eq_S   oilrfl_t:vr_w   oilrfl_p:dv

      #    The ‘state index’ that precedes the state values may be an
      #    integer variable or a string variable
      #
# state
# index     value             value                value

@STATE_VALUES_TABLE

   44       0       cmp_in_t:1[sec]       rail_p:122[psi]:4[sec]
   Twelve   TRUE    cmp_in_t:2[sec]       rail_p:22[psi]:8[sec]
```

Appendix C. Vertical Labels state_mon Specifications

```
@FILE_FORMAT
   VERTICAL_LABELS

      # the following defines the rate at which the variables are
      # monitored

@PROCESS_INTERVAL
   .5[sec]


      #     The ‘state index’ may be an integer variable or a string
      #      variable

# state
# index              index value         index value     index value

@STATE_INDICES
                           44                  12             five


@STATE_VALUES_TABLE

# variable:action        value                value               value


Engine_Run:EQ_S           0                 not_running            -

Safety_Trip:EQ           TRUE                 FALSE                -

oilrfl_t:lo_w       cmp_in_t:3[sec]          cmp_in_t:4[sec]       DC

oilrfl_p:dv_c       rail_p:122[psi]:4[sec]   "rail_p:22[psi]:3[sec]
-

clnt_sw:EQ_C            0                   1                    -

ecm_model:EQ_W           DC                    'cm870'
```

Appendix D. gp_test Keywords for state_mon

The following is an example of the keywords that should be placed in a test procedure mode when the CyFlex variables should be monitored. For this example:

- When `state_mon` indicates a normal or successful return, the next mode will be the default next mode.
- When `state_mon` indicates a failure return, the next mode will be mode `100`.
- The specification file is `/specs/gp/stbls/stbl_CLD1_tree` and it should be read only once at start up.
- When a `warning` variable failed to meet the specified state, the next mode will be the default next mode.
- When a `critical` variable failed to meet the specified state, the next mode will be mode `91`.
- When a `state_change` variable failed to meet the specified state, the next mode will be mode `110`.
- When an error occurs when reading the specification file, the next mode will be mode `105`.
- The group of specified states that should be used when monitoring the variables is specified in the CyFlex variable `CL1TreMdDsc`.
- The desired monitor action is specified in the CyFlex variable `CL1TreStMAct`.
- The maximum length of time to monitor the variables is specified in the variable `CL1TreMdTOt`.

```
@STATE_MON_ACTIONS
   #success_path     failure_return_path     read_mode   action_code
      MODE_TERMINATE        100              READ_ONCE   CL1TreStMAct

@STATE_MON_EXCEPTION_PATHS
# timeout timeout state change warning failure  critical failure read
error
     #     path       path          path             path        path
CL1TreMdT   90        110           91          MODE_TERMINATE    105

@STATE_MON_SPEC_FILES
   #file_list (16 maximum)               state_index
   /specs/gp/stbls/stbl_CLD1_tree        CL1TreMdDsc
```

Appendix E. gp_test Trace Information Generated by state_mon

Below is a example snippet of the diagnostic trace information generated when `state_mon` has been called from a `gp_test` mode. The basic information provided is as follows:

- The first line indicates the PID of the `state_mon` becoming active (`6735`), how the specification files should be read (`read_once`) and the number of specifications being read (5).
- The second line indicates the `MONITOR` operational mode for `state_mon`.
- Next is a heading that describes the information that will be written to the TRACE file.
- Following the heading is the value of the state index that was used to determine which group of values should be used when the specified variables are checked. The first specification file had an index value of `ok`.
- After the state index value is the information associated with each state variable being checked.

As indicated by the TRACE, all the variables have been 'verified' and they are now going to be 'monitored'. It is important to know the state of all the variables when `state_mon` is called; therefore, the information for all variables is written to the TRACE file when *state_mon* is initially called.

Looking at the first informational line written, the following information is provided:

- Time the variable was checked: `0.00`
- A number indicating which specification file specified under the *@STATE_MON*_SPEC_FILES keyword contains the state variable (`0`). This indicates that the first file specified is being used.
- The label of the variable being checked: `cah_frz_sts`
- The action associated with the variable: `EQ_S`
- Whether or not the possible action extensions for the variable passed the check, (`1,1,1,1`) The one that is relevant (state passed) is indicated by the specified action extension: _S. All others are defined to have passed.
- The actual value of the variable: `ok`
- The expected value of the variable as specified in the spec file: `1`

As expected, all the variables passed their checks when the mode was first executed. The test procedure stayed in the mode for `2908.12` seconds. At this time the variable `air_mf_lim` changed from a value of `ok`' to `low_flow`. When this happened, the mode was terminated and `gp_test` was signaled to take the `state_change` failure path.

```
---------------------- TRACE file snippet -----------------------------
.
.
gp_state_mon.cah,134  | State VERIFIED, now MONITOR.  Exit on state change.
 T/D-12:19:04 01/13/14

smCFG to 6735 action=MONITOR read_once=1 nfiles=5
```

```
state_mon - start of MONITOR mode - 01/13/14 12:19:04.30

   time:(file_num)    vrbl_name:state (crit,warn,state,vrbl)  actual,
expected
                                action (pass pass pass  pass)   value  value
     state_index = ok ---------
   time=  0.00:(0)         cah_frz_sts:EQ_S  (1, 1, 1, 1)           ok, 1
   time=  0.00:(0)          engine_run:EQ_S  (1, 1, 1, 1)  Don't Care Spec
   time=  0.00:(0)           cah_fault:EQ_S  (1, 1, 1, 1)           ok, 0
     state_index = enable_on ---------
   time=  0.00:(1)          cah_en_sts:EQ_S  (1, 1, 1, 1)           on, 1
   time=  0.00:(1)        cah_cold_lim:EQ_S  (1, 1, 1, 1)  Don't Care Spec
   time=  0.00:(1)         safe_status:EQ_S  (1, 1, 1, 1)         safe, safe
   time=  0.00:(1)          engine_run:EQ_S  (1, 1, 1, 1)  Don't Care Spec
     state_index = damp_open ---------
   time=  0.00:(2)      cah_sup_dmp_op:EQ_S  (1, 1, 1, 1)         open, 1
   time=  0.00:(2)       enable_status:EQ_S  (1, 1, 1, 1)           on, on
   time=  0.00:(2)          engine_run:EQ_S  (1, 1, 1, 1)  Don't Care Spec
     state_index = exitP_damp_open_okP ---------
   time=  0.00:(3)         cah_otP_lim:EQ_S  (1, 1, 1, 1)           ok, 1
   time=  0.00:(3)         damp_status:EQ_S  (1, 1, 1, 1)         open, open
   time=  0.00:(3)          engine_run:EQ_S  (1, 1, 1, 1)  Don't Care Spec
     state_index = flow_ok_run ---------
   time=  0.00:(4)          air_mf_lim:EQ_S  (1, 1, 1, 1)           ok, 1
   time=  0.00:(4)     pressure_status:EQ_S  (1, 1, 1, 1)          run, run
   time=  0.00:(4)          engine_run:EQ_S  (1, 1, 1, 1)      running, 1

     state_index = flow_ok_run ---------
   time=2908.12:(4)          air_mf_lim:EQ_S  (1, 1, 0, 1)     low_flow, 1

 T/D-13:07:32 01/13/14
   state_mon -  state_mon state change - took state_change exit path
specified
```