WHEN YOU NEED TO BE SURE

CyFlex® Common Test Manager Keywords

Version 8

May 31, 2022

Developed by SGS North America, Inc.

Version History

Version	Date	Revision Description
1	6/23/2016	Initial publication
2	12/6/2016	Remove obsolete `NO_RUN_PROCEDURE`
3	8/23/2018	Format with SGS brand
4	4/2/2020	Retrofit to new template Added Overview section
5	8/18/2020	Added the following sections : • Added *Section 12.3 @DYN and @THR* on page 24 • Added *Section 12.5 @SPD and @TRQ* on page 26
6	8/18/2021	Added *Section 2.8 @PATH_OPTIONS* keyword on page 5
7	12/20/2021	Removed mention of `AT_NT kill_option` from *Section 7.1 @BACKGROUND_TASK* on page 13
8	5/31/2002	Updated hypertext linked cross-reference to cyflex.com usage help for `floger` in *Section 9 Data Acquisition* on page 17

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

LIST OF TABLES

1 Overview

This document provides a consolidated reference for commonly used CyFlex Test Manager keywords. Refer to the following topics:

2 Mode Timing and Branching

2.1 @MODE

Use this keyword is to declare the mode number, maximum time for a test mode, the `default_next_mode` to execute when this mode is complete and a description of the mode which can be displayed on the test cell monitor.

The `@MODE` specification must be present for every mode. The `time_out` value may be expressed in any units of time. For instance, 60[sec], 1[min], .01667[hr] are all equivalent. A zero value indicates the mode has indefinite length and will be terminated by some means other than a simple time out. A negative number (-1[sec]) indicates that no time is to be spent in the mode other than to perform the actions specified in the mode. This is called an "immediate" mode, meaning perform the actions and immediately move on the next mode.

The `default_next_mode` is called the default mode because it may be overridden by other specifications which control the path. It defines the path taken when the mode is terminated and no other execution path applies. This path may be a mode number, `RETURN`, or a test procedure filename.

Use `RETURN` for the `default_next_mode` to return to a calling test procedure. Refer to *Section 2.6 @PROCEDURE* on page 4.

```
@MODE
#mode_number    time_out     default_next_mode
43              30.0[sec]     RETURN
#description
stay at this condition for 30 seconds
```

2.2 @LOOP_CONTROL

Use this keyword to specify a looping operation. Control will be passed to the `next_loop_mode` for the number of times specified in the `num_repeats` parameter. Then control is passed to the `default_next_mode.` An optional parameter, the `loop_counter_variable` can be used to specify an integer variable label which will track the number of times the loop has been executed. If a variable label is specified and does not exist, it will be created by `gp_test`. The `num_repeats` parameter may be a constant, variable label, or a computed expression.

```
@LOOP_CONTROL
   # num_repeats    next_loop_mode     loop_counter_variable
   10               51                 loop1_ct
```

2.3 @TEST_CYCLE_END

Use the keyword @TEST_CYCLE_END to designate the last test mode in a test cycle. This is used only when either of the following is a requirement:

- An accurate count of cycles must be maintained over a long period of time
- The test must be terminated after the required # of cycles

```
@TEST_CYCLE_END
   #maximum_cycles     counter_label    test_complete_path
   30000               test_cycles      91
```

When the number of cycles reaches the value specified for `maximum_cycles`, then control is passed to the test mode or procedure specified as `test_complete_path`. This would normally be a shutdown sequence. It may be specified as a mode number, procedure filename, or with RETURN, to return to a calling procedure.

The counting of test cycles is based on a user-specified CyFlex® integer variable and the Test Manager never initializes or resets this counter. It is the responsibility of the test cell operator to reset the counter with the `svar` command. This allows a test to be restarted several times and have the cycle counter accumulate the count of test cycles over a long period. This variable is continually updated on the hard disk so this count is not lost if the computer is powered down or the system is restarted.

2.4 @IF_TRUE & @IF_FALSE

The control and data acquisition actions of a mode will not be executed unless all of the conditional variable tests are satisfied. All of the logical variables or expressions listed under the @IF_TRUE keyword must be true and all of the logical variables or expressions listed under the @IF_FALSE keyword must be false. Otherwise, control is passed to the path specified with the @ELSE_MODE keyword. If no ELSE_MODE is specified, control is passed to the `next_mode`. The variables and expressions are evaluated only at the start of the mode.

```
@IF_TRUE
   #variable_label
   Engine_Run
   "RPM > 1000[rpm]"
@IF_FALSE
   #variable_label
   Safety_trip
   SpeedLT400
   "ave_exh_t < 1000[deg_f]"
```

2.5 @ELSE_MODE

Use the `else_mode` keyword to designate an alternate execution path. This mode or test procedure will be executed if the conditional tests fail. The macro RETURN may be used to cause control to be returned to a calling procedure or a procedure filename may be specified.

```
@ELSE_MODE
99
```

2.6 @PROCEDURE

Use the @PROCEDURE keyword to specify the file name of another complete test procedure file to be run as a test mode of the calling procedure. No other operations (keywords) should be specified in the calling mode with the exception of the conditional testing functions @IF_TRUE, @IF_FALSE, @SWITCH, and @ELSE_MODE.

```
@PROCEDURE
   # pathname
   /specs/gp/gp_subcycle
```

2.7 @SWITCH

The @SWITCH keyword provides a method for controlling the path of execution based on the value of an integer variable, string variable, or a computed value. The keyword is followed by the label of the variable or an expression to be computed, then by a list of (value, path) combinations which specify the path of execution for possible values of the variable or expression. If the resultant value or the variable/expression does not match a value in the list, then the current mode is executed; otherwise the path specified for that value is taken.

The path may be a mode, procedure filename or RETURN.

Example using an integer variable (myvar).

```
@SWITCH
#switch_variable
myvar
#value      path
1           43
2           95
3           /specs/gp/gp_shutdown
4           RETURN
99          1
```

Example using a string variable (myString).

```
@SWITCH
#switch_variable
myString
#value      path
'IDLE'      /specs/gp/gp_idle
'PAUSE'     95
'STOP'      /specs/gp/gp_shutdown
'COMPLETE'  RETURN
```

2.8 @PATH_OPTIONS

The @PATH_OPTIONS keyword provides a method for controlling the path of execution based on the evaluation of computed expressions which must resolve to a TRUE or FALSE value. If the result of the expression is TRUE, then control passes to the corresponding path for that option. The options are evaluated as “conditional” tests similar to @IF_TRUE, @IF_FALSE, and @SWITCH. The evaluation of this keyword is performed at the start of the test mode prior to execution of any other actions of the mode. Each of the options in the list are evaluated in the order in which they appear in the list. If any option evaluates to TRUE, the path of execution is modified, the mode terminates, and control is passed to the specified path. If none of the expressions evaluate to TRUE, then the current mode is executed.

The path may be a mode number, procedure filename, or RETURN.

```
@PATH_OPTIONS
   #expression              exit_path
   “my_logi && xyz”         100
   “my_logi && !abc”        101
   …etc… (up to 24 specifications)
```

Note:

The @PATH_OPTIONS keyword will be available in CyFlex.6.3.26 and later versions.

3 Mode Termination

Mode termination may be caused by the mode time-out timer, completion of a support operation such as:

- Data acquisition functions
- Completion of stabilization
- By receiving a user-specified event; refer to *Section 3.3 @TERMINATION_EVENTS on page 7*.
- By a special limit specification being violated which is unique to the test mode

3.1 @LIMIT_SPECS

Use the `@LIMIT_SPECS` keyword to specify a list of limits on real, `REAL_ARRAY`, `INTEGER`, `INTEGER_ARRAY`, `LOGICAL`, statistical, composition, and property variables which will cause the test mode to terminate if the limit is violated.

Table 1: @LIMIT_SPECS Data Fields

Data Fields	Explanation
`variable`	The limit variable label
`value`	The limit value the variable will be compared to. This may be expressed as a constant, variable label, or computed expression.
`type`	A character which designates whether the limit is an upper or lower limit. Options are: U - upper L – lower
`interval`	The rate at which the limit is to be examined Options are: `FAS`, `MED`, `SLO`, and `WARP`
`period_out`	A time interval during which the variable must continuously violate the limit value before action is taken.
`next_path`	The mode or test procedure to execute next if this limit is exceeded. A dash '-' may be used to indicate the normal next mode is to be used. The macro `RETURN` can be used to cause a return to the calling procedure.

The value field may be expressed as either a constant, variable label, or a computed expression. If the value is a constant, the units keyword must be appended as shown in the example below.

```
@LIMIT_SPECS
#variable   Upper/Lower        rate
# label    value      type  interval  period_out  next_path
my_temp    200[deg_F] U     SLO       3[sec]        _
RPM        2500[rpm]  U     MED       0[sec]     /specs/gp/gpx
```

3.2 @LIMIT_SPECS_ALL

@LIMIT_SPECS_ALL is similar to @LIMIT_SPECS except that all of the specifications must be violated for the termination path to be taken. This is like a logical AND of all the specifications, whereas the @LIMIT_SPECS keyword is like a logical OR. Unlike the @LIMIT_SPECS specification, only one exit path may be specified for the entire group. Thus, the format is slightly different with the exit path being specified immediately after the keyword and followed by a list of the specifications.

Table 2: @LIMIT_SPECS_ALL Data Fields

Data Fields	Explanation
variable	The limit variable label
value	The limit value the variable will be compared to. This may be expressed as a constant, variable label, or computed expression.
type	A character which designates whether the limit is an upper or lower limit. Options are: U - upper L - lower
interval	The rate at which the limit is to be examined Options are: FAS, MED, SLO, and WARP
period_out	A time interval during which the variable must continuously violate the limit value before action is taken.

Note that two string variables may be designated in the instance definition file (usually /specs/gp/gp_header) for displaying the list of limits which have are currently in violation and those which aren't. See the *Instance Definition File* section in the *Test Manager User Guide*. These variable names are usually defaulted to TrmLIM_done and TrmLIM_waiting.

3.3 @TERMINATION_EVENTS

The @TERMINATION_EVENTS keyword is followed by a list of event names which will terminate the mode. Use the next_path field to specify the test mode or test procedure to be executed. If next_path is a zero, dash ("-") or not specified, then the normal next mode will be executed. The macro RETURN may be used to cause a return to a calling procedure. It is not required that the events exist when the test is started. At each start of execution of the mode, an attempt will be made to get the event ID if it was not available at the start of the test.

```
@TERMINATION_EVENTS
    #event_name         next_path
    stop_this           90
    stop_that           91
    go_there            /specs/gp/gp_other
    proc_done           RETURN
```

3.4 @TERM_ALL_EVENTS

The @TERM_ALL_EVENTS keyword is followed by a list of event names which must ALL be received to cause termination of the mode. It is not required that the events exist when the test is started. At each start of execution of the mode, an attempt will be made to get the event ID if it was not available at the start of the test.

```
@TERM_ALL_EVENTS
   #event_name
   stop_this
   stop_that
   go_there
   proc_done
```

Note that two string variables may be designated in the instance definition file (usually `/specs/gp/gp_header`) for displaying the list of events which have been received and those which have not. See the *Instance Definition File* section in the *Test Manager User Guide*. These variable names are usually defaulted to `TrmEV_done` and `TrmEV_waiting`.

4 Setting Events

Use the @SET_EVENTS keyword to send synchronization signals (events) to start other processes. Up to 16 events may be listed per test mode. It is not required that the events exist when the test is started. At each start of execution of the mode, an attempt will be made to get the event id if it was not available at the start of the test.

```
@SET_EVENTS
   #list of event names which will be set at designated time
   #start_type          event_name
   AT_START             get_it_going
   AFTER_STABILITY      take_data
```

5 Using Stabilization Criteria

Various operations in a particular mode may be initiated after some stabilization criteria are satisfied. The operations which may be initiated are indicated by the STABILITY_ACTION macro. It is not necessary to specify a STABILITY_ACTION keyword if no special action is required. The operations specified to occur AFTER_STABILITY will still take place.

5.1 @STABILITY_ACTION

Use the @STABILITY_ACTION keyword to specify one or more special actions which indicate how the stability specifications are to be applied. At least one of the following must be specified if there are stability specifications:

- MODE_TERMINATE
- TERMINATE_TO_ELSE_MODE
- WAIT_FOR_STABILITY

Example:

```
@STABILITY_ACTION
    MODE_TERMINATE
```

5.2 @STABILITY_SPECS

The @STABILITY_SPECS specification is a list of the various criteria which are to be evaluated. Stability is complete when all of the specified criteria are achieved.

Note:

The reference field of a specification may be a constant, a variable label, or a computed expression.

Table 3 lists the types of available stability criteria.

Table 3: Available Stability Criteria Types

Type	Evaluation
TIME_DELAY	set a minimum time for stability
VARIANCE	maximum variation for time period
CURRENT_DEVIATION	maximum deviation from reference
DEVIATION	maximum deviation from reference for time period
STANDARD_DEVIATION	maximum standard deviation for time period
K_VARIANCE	maximum coefficient of variability for time period

The reference, tolerance, and min_ref fields may be expressed either as constants, variable labels, or computed expressions. If the field is a constant, the units keyword must be appended as shown in the examples on page 11.

```
@STABILITY_SPECS
#type_code   label   timeout   rate   reference   tolerance  min_ref
   TIME_DELAY      -   20[sec]
   VARIANCE   RPM      10[sec]  SLO    -          10[rpm]
   K_VARIANCE RPM      10[sec]  SLO    -            5[rpm] 1000[rpm]
   DEVIATION  TORQUE   10[sec]  SLO    Torq_Peak    5[none]
   VARIANCE   flotron   1[min]  SLO     -          2.0[lb/hr]
```

6 Setting Output Variables

The @PARAMETERS keyword allows up to 64 specifications to be entered for each test mode. Each parameter may be specified to be modified at the start of the mode, after stability, or at the end of the mode. The optional `restore_flag` can be used to designate that the value of the parameter is to be saved and restored to its original value when the test mode is complete.

The value field may be a constant, variable label, or a computed expression.

```
@PARAMETERS
#start_type     label      value/variable/expression      restore_flag
AT_START        bit_69     ON                               YES
AT_START        i_var      100[none]                        RESTORE
AT_START        x_var      99.9[psi]
AT_END          my_comp    "(RPM * 2[none] ) + 100[rpm]"
AT_START        my_logi    "if (RPM > 200[rpm]) then TRUE else FALSE"
```

7 Launching Background Applications

7.1 @BACKGROUND_TASK

Any Linux or CyFlex command, application, or script may be launched in the background. The execution can be performed at the start of the mode or after stabilization. There is no error checking or synchronization performed. Include the command and arguments in quotes. The `kill_option` field is optional and defaults to NONE. Use the `kill_option` to specify whether or not the spawned process should be slayed and when.

`kill_option`

Specify the following:

- NONE to indicate the process is never terminated by `gp_test`
- AT_START to terminate the process at the start of a mode if it was launched in the previous execution of the mode
- AT_END to terminate the process is when the mode is terminated

```
@BACKGROUND_TASK
 #start code  "background process task name"       kill_option
 AT_START     "/specs/cmds/myscript"
 AT_START     "/cyflex/bin/floger  /specs/logr_filname"
```

7.2 @BACKGROUND

This keyword is similar to @BACKGROUND_TASK except that the command string may be a constant (literal string), a string variable label containing the process name, or computed expression. This is the preferred and recommended keyword. @BACKGROUND_TASK will be eliminated in future revisions.

```
@BACKGROUND
   #start code  command_string                      kill_option
   AT_START     '/specs/cmds/myscript'
   AT_START     my_script
   AT_START       " '/cyflex/cmds/'  +  my_script  "  AT_END
```

7.3 @AUXILIARY_TASK

The @AUXILIARY_TASK keyword supports launching of some special support applications that are designed to synchronize with the Test Manager. They are launched with command line arguments which inform them of event names that are used to signal the start of operation, cancelling of operation, and a reply message that the support application uses to signal success or failure of its operation. There are only a few support applications designed to operate this way and those are the only ones that can be specified using this keyword. These include:

- `vrbl_to_file`
- `exec_script`
- `engine_start`

```
@AUXILIARY_TASK
  #start_type    success_path            failure_path
  AT_START       MODE_TERMINATE          99
  #task pathname            "command line"
  /asset/bin/vrbl_to_file  "/specs/gp/vrbls/myvrbl  READ  count"
```

Example with string variable containing the 'command line':

Note:

> The string variable `vrbl_args` is set to '`/specs/gp/vrbls/myvrbl  READ  count` in a previous mode.

```
@AUXILIARY_TASK
  #start_type    success_path            failure_path
  AT_START       MODE_TERMINATE          99
  #task pathname             "command line"
  /asset/bin/vrbl_to_file    vrbl_args
```

Example of using 'exec_script'

Note:

> The script `/specs/cmds/my_script` is a shell script that exits with a value of `0` (.i.e. `exit 0`) when it performs its function successfully or it exits with a non-zero value (.e.g. `exit 7`) when its function fails.

```
@AUXILIARY_TASK
  #start_type    success_path            failure_path
  AT_START       MODE_TERMINATE          99
  #task pathname             "command line"
  /asset/bin/exec_script   "/specs/cmds/my_script"
```

7.4 @TEST_TABLE

The @TEST_TABLE keyword is a special case of @AUXILIARY_TASK which causes the `vrbl_to_file` process to be launched to "READ" the spec file.

```
@TEST_TABLE
   #success_path          failure_path
   MODE_TERMINATE         99
   #filename              index_variable
   /specs/gp/tt1          count
```

Options for `success_path` and `failure_path`:

- `NONE`
- `MODE_TERMINATE`
- `RETURN` (to calling procedure)
- A mode number in this test procedure

The `filename` is the name of a `test_table` file. This may be a filename such as `/specs/gp/vrbls/test_tablex`, the label of a string variable which contains a file name, or a computed expression such as " `/specs/tt/ + filenumber` "

The `index_variable` is the label of an integer variable which is used to determine which record is read from the `test_table` file.

8 User Control Loops

User controls are any closed loop control other than the engine/dyno control loops. Typically, these are used for controlling the temperature or pressure of fuel, coolant, air, etc. The @USER_LOOP specification is used to change the target for a control loop which has already been established in the `/specs/ctrl_specs` file.

If the open loop start and end targets are different, the target will be ramped linearly based on the mode timeout value.

The target fields and ramp rate may be expressed as either constants, variable labels, or computed expressions. If the target is a constant, the units keyword must be appended as shown in the examples below.

The ramp rate has an implied "per second" rate.

```
@USER_LOOP
#open/closed variable  start_target  [end_target]  [ramp_rate]
CLOSED_LOOP  port_in_p 150[deg_F]
CLOSED_LOOP  fuel_temp 104[deg_F]
OPEN_LOOP    air_inlet_t 100[%]
CLOSED_LOOP  the_temp  100[deg_F]  My_target_t
CLOSED_LOOP  the_pres  0[in_H2O]   "my_target_p + 5.0[in_H2O]"
```

9 Data Acquisition

Several types of data acquisition processes may be controlled through procedure file specifications. In most cases, a separate task handles the data collection process. The specifications are handed off to the collector task and it is told when to start or stop. The collector task will usually reply when the process is complete. The user has the option to terminate the test mode at that time.

The following data acquisition processes are supported:

- Test data logger (spawned as a background task)
- Formatted writes to a file
- Fuel readings or PAM datapoints, refer to *Section 13 PAM Fuel Reading Data Acquisition* on page 29.

The fast data logger, `floger`, may be managed through a test procedure in 3 different ways.

1. Setting `floger` control events for an instance of `floger` launched in the `go.scp` startup script,
2. Launching an instance of `floger` as a supporting application, or
3. Launching an instance of `floger` as a background application

The first two methods are the preferred methods, depending on the user's requirements. Refer to cyflex.com usage help for `floger` for related information.

The first method is used when the data logger is assumed to always be present and only needs to be started and stopped based on the test procedure sequence. The @SET_EVENT keyword can then be used to set the start and stop events.

The second method is used when the data logger is associated with a particular test procedure and doesn't need to be present unless that procedure is running. In this case the data logger should be launched from the root procedure using the @CO_PROCESS keyword in the global definitions section of the procedure file. The global definitions section is after the `start_mode` specification, but before the first @MODE keyword.

The third method can be used if a data logger is only needed for a particular procedure but can be left running in the system even after the test procedure is no long being run by this instance of the Test Manager. Normally this is not useful, but a test procedure could also send a 'release' event to a data logger, causing it to terminate just before the test procedure itself terminates.

9.1 @WRITE_VALUES

The @WRITE_VALUES operation allows the user to write text data into a file and control the data, format, and rate through the test procedure. A data logging type of operation may be created through the Test Manager. The most common use is to capture the value of a particular variable after the operating conditions have been obtained through the test procedure. In order to maintain real-time operating performance, the actual disk I/O is handed off to a support application called `fw_coll` so that the `gp_test` application cannot be held up by disk or file system delays.

Table 4: @WRITE_VALUES Data Fields

Data Fields	Explanation

`start_type`	When to write the data.
	Options are:
	`AT_START`
	`AFTER_STABILITY`
	`AT_END`
	`NEW_FILE – remove file and re-open`

`file_name`	The file where the data will be written. This may be expressed as a literal pathname such as '/data/PC_format/myfile' or the label of a string variable may be entered. The string variable must then contain the pathname.
`label`	The CyFlex variable for which the value is to be obtained. A dash "-" indicates no value/variable. Only one label may be used per line in the specification
`"format_string"`	The C format string to be used for formatting the write. Quotes are required. Use %d for integers, %s for strings, and n.m%f for floating point variables such as real, statistical, arrays, etc.

Examples: (up to 8 specifications per mode)

```
@WRITE_VALUES
   #start_type  file_name       label   "format_string"
   AT_START     /data/tq_sp     -       "rpm "
   AT_START     /data/tq_sp     -       "torque \n"
   AT_START     /data/tq_sp     RPM     "%11.2f "
   AT_START     /data/tq_sp     TORQUE  "%11.2f "
```

10 Ramping

The Test Manager can launch a support application that will output a sequence of values to a variable resulting in a linear ramping of the variable from a start value to an end value with output updates at a specified rate.

10.1 @RAMP

```
@RAMP
   #variable    start        end          rate         termination
   my_var       x            y            z            AT_END
```

- `variable` is the label of the target variable.
- `start` is an optional start target (constant/variable/expression). If the `start` target is entered as a dash, then the start target is taken from the current value
- `end` is an optional end target (constant/variable/expression). The end target may be entered as a dash only for the case where the termination option is `MODAL` (default). If the termination option is `NONE`, then the end target must be specified
- `rate` is an optional ramp rate (constant/variable/expression). If the rate is not specified or is entered as a dash '-', then it is computed from the start/end and mode time. The rate is implied as [units]/sec.
- `termination` is an optional termination mode. The default is `AT_END`.
  - Specify `AT_END` to terminate the ramping operation when the mode terminates. The last value is frozen at the value reached by ramping and is not stepped to the end target.
  - Specify `NONE` to continue the ramping operation until the end target is reached even if the test mode terminates.

Notes:

- The `start`, `end`, and `rate` values are determined only once, when the ramping operation is started.
- Use of a variable or expression could result in a ramping operation running indefinitely.
- Specify up to a total of 8 ramps in any mode. This limit applies to the total of all @RAMP specs and all `@RAMP_DYNAMIC` specs.
- There may only be a total of 16 ramping operations going on at the same time. This will include the sum of all ramps generated with `NONE` termination codes in previous test modes, plus all ramps running in the current mode.

10.2 @RAMP_DYNAMIC

This functions identically to the @RAMP capability except that the end and rate are dynamically re-evaluated every FAS interval until the target is reached.

```
@RAMP_DYNAMIC
   #variable    start        end          rate         termination
   yore_var     ex           why          zee
```

11 Email/Paging/Texting

An email, page or text message may be generated by a test procedure. This is usually to inform someone of the progress of a test or that something has gone wrong with it, such as an unexpected shutdown.

The specification requires an `address` and a `message`.

- `message` is the message content. It may be a short string such as `test done in tc115` or can be derived from a file.
- `address` may be any email address such as:
  `Joe_Engineer@companyname.com`
  Also, the string "NOTES" in an email address will be replaced by "companyname.com" to save space.
- `subject` is an optional field which may be a single-quoted string, a double-quoted expression, or the label of a string variable which contains the subject. If the subject field is omitted, the subject will be `gp_test_EMAIL`.

The following paging systems are supported with email type domain names.

- Company 8800 Pager System
  `xxxx.companyname@assetpager.facility.companyname.com` (where `xxxx` is the pager number)

  The message should be a numerical string that can be displayed by the pager such as `115`.
- Indiana Paging Network
  `xxxxxxxxxx.indiana@assetpager.facility.companyname.com`
- Skytel
  `xxxxxxx.skytel@assetpager.facility.companyname.com`
- Verizon cell phones and pagers
  For text message: `xxxxxxxxxx@vtext.com`
  For pagers: `xxxxxxxxxx@vtext`
  where `xxxxxxxxxx` is the phone or pager number

There may be up to four separate specifications per test mode.

```
@EMAIL
  #start_code  receiver                message                subject(optional)
  AT_START     'Ryan.Lauer@NOTES'      ‘help, I have fallen’ ‘down and out'
  AT_START     user_email              /specs/canned_msg
  AT_START     /specs/email_list       /ram/msg
  AT_END       ‘8125551212@vtext.com’  ‘gp_map’               ‘test done’
```

12 Engine Control

12.1 @ENGINE_CONTROL_MODE

The control mode is used to select a dyno and throttle control combination which will be sent as a configuration message to the closed loop control application. The normal mode is to have the throttle control gross torque and the dyno control speed (control mode 4). For "no-load" operation, the throttle must be used to control speed (control mode 2).

The engine control mode does not have to be set unless it is being changed from mode to mode.

Table 5 shows the available control modes.

Table 5: Control Modes

Dyno controlling	Throttle controlling	control_mode
Dyno-torque	Speed	1
Net-torque	Speed	2
Other parameter	Speed	3
Speed	Gross-torque	4
Speed	Net-torque	5
Speed	Pther parameter	6

Example:

```
@ENGINE_CONTROL_MODE
4
```

A macro string may be substituted for the control_mode number as shown in *Table 6*.

Table 6: Macro String Substitutions

control_mode	string
1	DYNO_DYNO_TORQUE_THROT_SPEED
2	DYNO_NET_TORQUE_THOT_SPEED
3	DYNO_OTHER_THROT_SPEED
4	DYNO_SPEED_THROT_GROSS_TORQUE
5	DYNO_SPEED_THROT_NET_TORQUE
6	DYNO_SPEED_THROT_OTHER

12.2 @DYNO and @THROTTLE

The @DYNO and @THROTTLE keywords are used to specify the mode of the controllers and the open loop target, if the control mode is open-loop. The target may be expressed as a constant, variable label, or a computed expression. All constants must have the units specified.

The macros OPEN_LOOP and CLOSED_LOOP are used to specify the control mode.

In open loop mode a single target may be specified or a start target, end target, and ramp_rate may be entered. If only the start target is specified, then the output position is set immediately. If a start and end target are specified, then the output is ramped from the start to the end target over the time interval specified as the time-out in the @MODE specification. If start, end, and ramp_rate are all entered, then the output is ramped from the start target at the specified rate until the end target is reached or until a new target is specified.

Open loop targets should be expressed in percent of full scale. Ramp rates are also expressed in % with an implied rate of (%/second).

It is important to note that these commands take at least 50 milliseconds to complete when using the old ctrl_task program because of internal delays These delays **do not** exist in the new eng_ctrl_task engine control program and new user_ctrl_task user control program available in CyFlex version 6.3. See *Section 12.3 @DYN and @THR* on page 24 for more information.

Examples:

```
@DYNO
# open/closed_loop   start_target end_targetramp_rate
OPEN_LOOP            0[%]
```

The preceding is a dyno in open loop mode set to zero excitation.

```
@DYNO
# open/closed_loop   start_target end_targetramp_rate
CLOSED_LOOP
```

The preceding is a dyno in closed loop mode (usually, the dyno will be controlling speed)

```
@THROTTLE
# open/closed_loop    start_target     end_target       ramp_rate
OPEN_LOOP             0[%]             100[%]          10[%]
```

The preceding is a throttle in open loop mode ramping from closed throttle to fully open throttle at 10% per second.

```
@THROTTLE
# open/closed_loop    start_target     end_target      ramp_rate
OPEN_LOOP             min_thr          "(max_thr-min_thr) /
2.0[none]"
```

The preceding is the throttle ramping based on a variable and a computed expression.

12.3 @DYN and @THR

Before the time of very high-speed dynamometers, the time delays in the @DYNO and @THROTTLE keywords were not considered restrictive for running engine tests. However, when response times became critical in order to pass EPA regression testing, these delays became impossible to overcome and needed to be removed. At that time, the @DYN and @THR keywords were introduced into the Test Manager. These keywords bypass the delays associated with communication between the Test Manager and the old `ctrl_task` control program. In order to accommodate backward compatibility with existing test procedures, both the long and short form of the commands are supported.

Use the @DYN and @THR keywords to specify the mode of the controllers and the open loop target if the control mode is open-loop. The target may be expressed as a constant, variable label, or a computed expression. All constants must have the units specified.

Use the macros `OPEN_LOOP` and `CLOSED_LOOP` to specify the control mode.

In open loop mode a single target may be specified or a start target, end target, and `ramp_rate` may be entered. If only the start target is specified, then the output position is immediately set. If a start and end target are specified, then the output is ramped from the start to the end target over the time interval specified as the time-out in the @MODE specification. If start, end, and `ramp_rate` are all entered, then the output is ramped from the start target at the specified rate until the end target is reached or until a new target is specified.

Open loop targets should be expressed in percent of full scale. Ramp rates are also expressed in % with an implied rate of (%/second).

Examples:

```
@DYN
# open/closed_loop   start_target end_targetramp_rate
OPEN_LOOP            0[%]
```

The preceding shows the dyno in open loop mode set to zero excitation.

```
@THR
# open/closed_loop   start_target end_targetramp_rate
CLOSED_LOOP
```

The preceding is a throttle in closed loop mode.

```
@THR
# open/closed_loop    start_target     end_target       ramp_rate
OPEN_LOOP             0[%]            100[%]          10[%]
```

The preceding is a throttle in open loop mode ramping from closed throttle to fully open throttle at 10% per second.

```
@THR
# open/closed_loop    start_target     end_target      ramp_rate
OPEN_LOOP             min_thr       "(max_thr-min_thr) / 2.0[none]"
```

The preceding is the throttle ramping based on a variable and a computed expression.

12.4 @SPEED, @TORQUE, and @OTHER_CTRL_VAR

The @SPEED, @TORQUE and @OTHER_CTRL_VAR keywords are used to specify the targets of the dyno and throttle controllers. The targets are used for control only if the control mode of the controller is *closed-loop*. The target may be expressed as a constant, variable label, or a computed expression. All constants must have the units specified.

There are three fields which can be entered. Only the first field, the `start_target`, is required. If the `end_target` is not entered, it is assumed to be the same as the `start_target`. If the third field, the `ramp_rate` is entered, then the target is ramped from the start to the end target at the specified ramp rate. If the start and end targets are different and the `ramp_rate` is not entered, then the target will be ramped linearly based on the mode timeout value.

Examples:

```
@SPEED
   #start_speed     [end_speed]     [ramp_rate(units/sec)]
   1200[rpm]
@TORQUE
   # start_torque   [end_torque]    [ramp_rate(units/sec)]
   1000[lb_ft]
@OTHER_CTRL_VAR
   # start_target   [end_target]    [ramp_rate(units/sec)]
   temp_target      "temp_target + 100[dt_F]"
@SPEED
   # ramp from 1200 rpm to 1400 rpm over the mode interval
   # start_speed       [end_speed]     [ramp_rate(units/sec)]
   1200[rpm]           1400[rpm]
@TORQUE
   # ramp to 1000 lb_ft at 100 lb_ft/second
   # start_torque      [end_torque]    [ramp_rate(units/sec)]
   1000[lb_ft]         1000[lb_ft]     100[lb_ft]
```

Important:

It is important to note that these commands take at least 100 milliseconds to complete when using the old `ctrl_task` program because of internal delays. These delays **do not** exist in the new `eng_ctrl_task` engine control program and new `user_ctrl_task` user control program available in CyFlex version 6.3. See *Section 12.5 @SPD and @TRQ* on page 26 for more information.

12.5 @SPD and @TRQ

Before the time of very high-speed dynamometers, the time delays in the @SPEED and @TORQUE keywords were not considered restrictive for running engine tests. However, when response times became critical in order to pass EPA regression testing, these delays became impossible to overcome and needed to be removed. At that time, the @SPD and @TRQ keywords were introduced into the Test Manager. These keywords bypass the delays associated with communication between the Test Manager and the old `ctrl_task` control program. In order to accommodate backward compatibility with existing test procedures, both the long and short form of the commands are supported.

Use the @SPD and @TRQ keywords to specify the targets for speed and load control loops. The targets are used for control only if the control mode of the controller is *closed-loop*. The target may be expressed as a constant, variable label, or a computed expression. All constants must have the units specified.

There are three fields which can be entered. Only the first field, the `start_target`, is required. If the `end_target` is not entered, it is assumed to be the same as the `start_target`. If the third field, the `ramp_rate` is entered, then the target is ramped from the start to the end target at the specified ramp rate. If the start and end targets are different and the `ramp_rate` is not entered, then the target will be ramped linearly based on the mode timeout value.

It must be noted that the speed and torque loops are not automatically set to closed loop when using the short form keywords. If closed loop operation is required, these keywords must be accompanied with the short form keywords @DYN and/or @THR, see . *Section 12.3 @DYN and @THR* on page 24.

Examples:

```
@SPD
   #start_speed     [end_speed]     [ramp_rate(units/sec)]
   1200[rpm]
@TRQ
   # start_torque   [end_torque]    [ramp_rate(units/sec)]
   1000[lb_ft]
@SPD
   # ramp from 1200 rpm to 1400 rpm over the mode interval
   # start_speed       [end_speed]     [ramp_rate(units/sec)]
   1200[rpm]           1400[rpm]
@TRQ
   # ramp to 1000 lb_ft at 100 lb_ft/second
   # start_torque      [end_torque]    [ramp_rate(units/sec)]
   1000[lb_ft]         1000[lb_ft]     100[lb_ft]
```

12.6 @PID_GAINS

Use the @PID_GAINS keyword to modify gain of any control loop. The gain change will remain in effect even when the current mode is terminated, however, a permanent change to the gain specifications is not made in the /specs/ctrl_specs.*NNN* specification file. The gains may be specified as a constant, variable, or computed expression.

```
@PID_GAINS
#loop            proportional          integral       derivative
DYNO_SPEED_GAINS    -.1[none]        -.01[none]       0.0[none]
Int_mnf_t           .01[none]         .001[none]      0.0[none]
```

The “loop” field is used to specify the particular closed loop controller for which the gains are to be modified. For “user” control loops this field specifies the controller name, which is usually the same as the label of the feedback variable. For engine controls, the field must be one of the following:

```
DYNO_SPEED_GAINS
DYNO_DYNO_TORQUE_GAINS
DYNO_NET_TOQRUE_GAINS
DYNO_OTHER_GAINS
THROT_GROSS_TORQUE_GAINS
THROT_NET_TORQUE_GAINS
THROT_SPEED_GAINS
THROT_OTHER_GAINS
```

For dual loop (hydraulic) dynos, use one of the following:

```
DYNO2_SPEED_GAINS
DYNO2_DYNO_TORQUE_GAINS
DYNO2_NET_TOQRUE_GAINS
DYNO2_OTHER_GAINS
```

12.7 @GET_PID_GAINS

Use the @GET_PID_GAINS keyword to retrieve the current value of the gains and to place the values into specified real variables. The real variables must already exist and have been created elsewhere (usually a gen_labels.* file).

```
@GET_PID_GAINS
#loop            proportional        integral       derivative
DYNO_SPEED_GAINS dyno_pg             dyno_ig        dyno_dg
Int_mnf_t        mnf_t_pg            mnf_t_ig
```

In the preceding specification, the proportional gain for the dyno controlling speed would be placed into the dyno_pg variable, and so on.

This keyword can be used to store gains so that they can be changed and later reset to their original values.

This keyword is not supported with the new engine control program because the present gain values are always available as CyFlex variables. They can be addressed directly.

12.8 @FEED_FORWARD

The feed-forward characteristics of a controller may be modified in any test mode. Feed-forward may be turned on or off. The feed-forward variable can be selected and the gain parameter can be specified. The gain parameter may be specified as a constant, variable, or computed expression.

```
@FEED_FORWARD
   #loop             active_flag      FF_label          gain
   DYNO_CTRLER       ON               power             .4[none]
   THROT_CTRLER      OFF
   int_mnf_t         ON               power            .01[none]
```

12.9 @GET_FF_GAIN

Use the `@GET_FF_GAIN` keyword to retrieve the current value of the feed-forward gain. The value is placed into the specified real variable. The real variables must already exist and have been created elsewhere (usually a `gen_labels.*` file).

```
@GET_FF_GAIN
   #loop             gain_label
   DYNO_CTRLER       dyno_ff_gain
   THROT_CTRLER      throt_ff_gain
   int_mnf_t         mnf_t_ff_gain
```

This keyword can be used to store the feed-forward gains so that they can be changed and later reset to their original values.

12.10 @CONTROL_TOLERANCE

The tolerance of a control variable can be specified. The value is not modified when the mode terminates. The tolerance may be specified as a constant, variable, or computed expression.

```
@CONTROL_TOLERANCE
   #variable         tolerance[units]
   SPEED_VAR         50[rpm]
   TORQUE_VAR        10[lb_ft]
    int_man_t         5[deg_F]
```

12.11 @OUT_CHAN_CONFIG

This keyword is used to modify the bias and gain of an output control channel for either engine controls or user loops.

See the `ctrl_specs.`*`NNN`* file for the current configuration. Note that this is a temporary change and will be reset after a “go”.

```
@OUT_CHAN_CONFIG
   #type source   chan/out_label bias     span/gain      filter
   AO   Dyno_CM      17          0        100            0
   RV   pp_cyl_p_CM  prbs_out     0        2              0
```

13 PAM Fuel Reading Data Acquisition

13.1 @FUEL_READING , @FUEL_READING_STATS and @PAM_DATAPOINT

These 3 keywords are used only for customer engine test systems. PAM is an acronym for the Performance and Analysis Module and database.

The fuel reading keywords are used to request fuel readings be taken during the test mode. A fuel reading collector process performs the function of requesting, timing, synchronizing, and counting the readings. When the process is complete, the collector sends a message to the Test Manager to indicate that the process is complete. If the MODE_TERMINATE option has been specified for the stop_path, the mode is terminated at that time.

The @PAM_DATAPOINT keyword allows a PAM datapoint to be captured without actually taking a fuel reading. The input format is identical to @FUEL_READING. The @FUEL_READING_STATS keyword contains additional data fields to specify statistical criteria for the confidence levels of the data.

A particular test mode may contain only one @FUEL_READING,@FUEL_READING_STATS, or @PAM_DATAPOINT keyword.

Table 7: PAM Fuel Reading Data Acquisition Data Fields

Data Fields	Explanation
start_type	The start_type determines when the fuel reading(s) begins. The collection process may be initiated immediately upon starting the mode (AT_START option) or after stabilization criteria have been satisfied (AFTER_STABILITY option). If the initiation of readings is to be controlled by an external event (EXTERNAL_SYNC option) then the readings are immediately enabled when the mode starts. Options are: AT_START EXTERNAL_SYNC AFTER_STABILITY
stop_path	The stop_path determines what happens when all of the fuel readings are complete. Options are NONE, MODE_TERMINATE, RETURN, a mode number, or procedure filename.
number_of_ readings	The number of fuel readings to be taken at this mode. This may be a constant, variable label, or computed expression.
interval	The time between fuel readings. This is the interval between each fuel reading start time and should therefore be larger than the expected fuel reading time. This field is ignored if the number_of_readings is one. If a non-zero value for interval is entered and the number of readings is greater than 1, then the readings are scheduled to occur at the interval rate. Units are required.

Data Fields	Explanation
sync_event	This field is used to specify an external event which will trigger the start of fuel readings. The event name is used only when the start_type is EXTERNAL_SYNC and should be entered as a dash, "-" for other start types.
desired_time	This is the target fuel reading time. For fuel scales, this becomes the target time for computation of the selected weight. For flotrons or other continuously measuring devices, it is used as the exact fuel reading time. Units are required. This field may be a constant, variable label, or computed expression.

Example specification (2 lines):

```
@FUEL_READING
   #start_type          stop_path
   AFTER_STABILITY      MODE_TERMINATE
   #number_of_reading   interval    sync_event    desired_time
      1                  0.00[sec]   -            2.5[min]
```

13.2 @PAM_GROUP_LIST

The Customer PAM database allows groups of datapoints to be "named" and then retrieved by the group name. For example, all of the datapoints collected at rated speed might be named 1800rated or repeat datapoints collected for quality assurance purposes might be named quality. This can simplify the extraction of a set of data.

Use the @PAM_GROUP_LIST keyword to automatically generate a name or several (up to 9) names for a particular fuel reading or set of fuel readings by placing PAM_GROUP_LIST in the same mode as the @FUEL_READING keyword. The datapoint file for readings taken in this mode will then include a PAM "GROUPS keyword with the group names assigned to that keyword. There is a mechanism in the PAM data extraction process which allows searching of datapoints to be based on a particular group name.

Nine string variable names have been reserved to stored group names. They are PAM_grp_1 through PAM_grp_9.

Use the @PAM_GROUP_LIST keyword in the same mode where the fuel reading is taken since the evaluation of computed expressions will be done at the mode start. There may be up to 9 group names assigned to a datapoint. The specification may be the label of a string variable, a literal string enclosed with single quotes) or a computed expression (enclosed with double quotes). Enter the different group names on 1 or more lines.

The group name for SPC or CLS datapoints should start with spc_ or cls_, respectively.

```
@PAM_GROUP_LIST
   #list of string variables for PAM group name(s)
   #( 1 or more lines - up to 9 labels per line)
   #the entry may be a label, literal string, or computed string
   PAM_grp_1   PAM_grp_2   PAM_grp_3   my_group
   'groupx'   "PAM_grp_1 + test_mode"
```

Group names in PAM are limited to 25 characters, even though a string variable may be 80 characters long. Only the first 25 characters of the string will be used for the group name.

This function provides the ability to synchronize several processes that are required to generate a PAM datapoint. The keyword allows the construction of a chain of events that provide the synchronization.

13.3 @FUEL_READING_SYNC

This keyword allows multiple processes to be synchronized with fuel readings when multiple fuel readings have been requested in a mode. The synchronization is handled externally from `gp_test`. The specification consists of a list of output events that will be emitted in the sequence that they are listed. Each output event is emitted when all of the input events listed on its line and all preceding lines have been received. This condition is overridden by the specified timeout (`0` timeout indicates no timer). The timeout for a particular line does not start until the output event on the previous line has been emitted. All input events are attached at the time a fuel reading is requested, so if an input event of a later specification line is received before those of a preceding line, it is still considered to be satisfied, but the corresponding output event would not be emitted until all those preceding it have been emitted.

Note:

The maximum specified delay for this entire process is the value of the variable `FR_wrte_delay`. If that time expires after the issuance `fr_ready`, the datapoint will be written even if `fr_write_ok` is not received. For a better understanding of the variables and events associated with fuel readings, refer to Gazette.6b.97-Variables, Events, and Processes associated with fuel readings

Table 8: @FUEL_READING_SYNC Data Fields

Data Fields	Explanation
`timeout`	Maximum wait time for the specified input events - the output event is issued if this timeout expires before all of the input events are received.
`output_event`	An event that will be set when all of the specified input events are received or the timeout expires
`input_events`	Up to 4 input events which must all be received before this sequence in the chain is satisfied.

Example Specification:

```
@FUEL_READING_SYNC
#when all the input events have arrived, the output event is emitted
#and we go to the next spec. Keep doing that until the list is
#complete

#event_sync (event sequences required to complete a datapoint)
#max_timeout       output_event       input_event_list (up to 4)
  0[sec]            TS_StrtAcq          fr_ave_strt
  0[sec]           TS_OpCondCmp         HS_AcqInPrg fr_ready HS_AcqCmp
  0[sec]               fr_write_ok      HS_AnlsCmp
```

Notes:

- `Fr_write_ok` should always be the last output event.
- The `FR_write_delay` is automatically set to 4 minutes when `@FUEL_READING_SYNC` is used.
- `@FUEL_READING_SYNC` can only be used in modes where `@FUEL_READING` or `@FUEL_READING_STATS` are also used.

13.4 @FR_LOG_FILE

Use the `@FR_LOG_FILE` keyword to define the data file that will be used to log fuel reading data in a columnar format that is compatible with spreadsheets.

Logging of fuel reading data can be turned on and off by setting the state of the variable `fr_log_enab`. This data logging operation is entirely separate from PAM data files and transfers.

Use the following syntax to enable logging:

```
set fr_log_enab ON
```

Table 9: @FR_LOG_FILE Data Field

Data Fields	Explanation
`file_pathname`	the full pathname of the data file

Example Specification:

```
@FR_LOG_FILE
       #file_pathname
       /data/fuel_log/fr_fuel_data
```

Log fuel reading data is placed in the file `/data/fuel_log/fr_fuel_data` if the `fr_log_enab` flag is `ON`.