CyFlex® Event Response Utility

Version 7

February 5, 2024

Developed by Transportation Laboratories

Version History

Version	Date	Revision Description
1	1/25/2016	Initial publication
2	8/23/2018	Format revisions
3	3/30/2020	Retrofit to new template
4	8/19/2021	Removed usage content *in Section 2 Event Response Commands* on page 4 and replaced with hyperlinked cross-references to cyflex.com usage help.
5	12/20/2021	Added descriptions of `@ELOG`, `@FAIL_ELOG`, and `@PASS_ELOG` keywords to *Table 2* on page 5 Added `@ELOG` entry *to Section 3.3 Example Specifications* on page 10
6	5/18/2022	Updated all hypertext linked cross-references to cyflex.com usage help descriptions
7	2/5/2024	Rebrand to TRP Laboratories

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

CyFlex's event response utility, invoked by the `evnt_rsp` command, allows test system administrators and users to define actions to execute when CyFlex events occur. The response to events which may be set up include:

- Determining a PASS or `FAIL` state based on CyFlex variable values or expressions
- Placing a message on a display console. Refer to *Section 1.2 Display Status* on page 2.
- Setting variable values
- Setting variable values with a one-shot time delay. Refer to *Section 1.5 One Shot Delay* on page 3.
- Changing the display status of a variable to make the display blink or change color
- Setting events
- Setting events with a one-shot time delay
- Logging an error or informational message
- Running a CyFlex command or script file
- Sending an email message

Many of the features that can be performed in a `gen_labels.`*`NNN`* specification file can also be done by the `evnt_resp` task by creating a specification in `er_specs.`*`NNN`*. However, there are some significant differences. The following list of `evnt_resp` features cannot be performed by a `gen_labels.`*`NNN`* specification.

- Setting of an event in response to an input event
- Changing the display status of a variable or several variables
- A single input event can cause the value of several output variables to be changed.
- Any one of up to 4 events can be specified to produce the same result.

On the other hand, an `er_specs.`*`NNN`* specification cannot be used to create a variable. This is usually done in `gen_labels.`*`NNN`*.

Up to 100 specifications may be entered in an `er_specs.`*`NNN`* file.

Multiple instances of the `evnt_rsp` task may be used, each with a separate specification file. Running multiple instances allows the specifications to be logically segmented for easier maintenance. For instance, specifications which are related to test cell systems can be handled by one instance and the specifications which relate to engine operation can be handled by another instance.

Refer to *Section 2 Event Response Commands* on page 4 for additional information.

1.1 CyFlex Variables and Events Concepts

To effectively configure and use the event response utility requires an understanding of the concepts of processes, variables, and events.

This section provides an overview of these concepts. Refer to the following for details:

- *CyFlex Variables, Units and Computed Expressions*
- *CyFlex Events*

1.1.1 CyFlex Variable

A CyFlex variable is the basic unit of data in CyFlex. There are several types of variables, each of which contains a different type of data as listed in *Table 1*:

Table 1: Summary of CyFlex Variables

Type of Variable	Description
Real Variables	used for analog inputs, computed performance variables, controls, clocks, and other parameters which have a continuum of values.
Integer Variables	used for counters, modes, indices, groups of digital inputs and outputs, and other parameters which have integer values.
Logical Variables	used for single digital inputs and outputs, flags, computed logical expressions, status and, other parameters that have two values (ON/OFF - TRUE/FALSE, etc.).
String Variables	contain a text string up to 80 characters long
Statistical Variables	contain several statistical values from a sampling operation
Composition Variables	contain several values defining the chemical composition of a fluid stream in the engine
Property Variables	contain several values defining the physical properties of a fluid stream at a particular temperature and pressure

1.1.2 CyFlex Event

A CyFlex event is a mechanism used to send a signal or data from one process to another within CyFlex. Events may be associated with the transition of a logical variable, i.e. a change in input or output value, change in test article or cell status, etc. They may also be set and processed as a result of other events or the passage of time. The term "trigger event" is used when discussing the event response utility to indicate an input event which causes a set of actions to be taken.

The event response utility is not capable of processing the data which is contained within some CyFlex events. Care should be taken when generating events. Typically, only events which are not required to contain data should be sent to event response. These are transition events associated with logical variables, starting another service like the data logger or statistics, etc.

1.2 Display Status

Each CyFlex variable, in addition to having a value, has a display status. The display status is used to change the display characteristics of a variable so attention is drawn to it. The display status value can change the color of the displayed variable and/or make it blink.

1.3 Keyword

A keyword is a text string which indicates the meaning of the information which follows. Event response uses keywords which are all capitalized and are preceded by an "at" sign (@). For example, the keyword **`@INPUT_EVENT`** specifies the trigger event for a group of actions. Any individual action or condition keyword may be used only once for each trigger event specification.

Keywords can be classified into the following different types:

1. Input or trigger: the start of a specification defining the input events.
2. Conditional test: logical variables or expressions which determine a PASS or `FAIL` state.
3. PASS actions: actions to execute when the conditional tests PASS.
4. `FAIL` actions: actions to execute when the conditional tests `FAIL`.
5. ALWAYS actions: actions to execute regardless of the conditional tests.

1.4 Commands and Scripts

A command or script is the name of an application or script file that is to be launched in the background. The `@PASS_SCRIPT`, `@FAIL_SCRIPT`, and `@SCRIPT` keywords are used to launch these applications. The file must be located in one of the normal executable paths such as `/cyflex/bin/`, `/cyflex/cmds/`, `/specs/cmds/` or else it must be specified with the complete pathname.

1.5 One Shot Delay

A one-shot delay is used to specify that the action is not to occur until a specified time after the input event has been received. The specification file is optional and if left blank, it is the same as a zero delay. In other works that action is executed immediately when the event is received. Until the specified delay has expired, the action is said to be “pending”. If the pending action is the modification of a variable value or status, and another event occurs before the one-shot delay has expired, then the pending action is cancelled and the more recent action takes precedence.

2 Event Response Commands

Two CyFlex processes provide event response capability:

1. The first, `evnt_rsp`, is a permanent application started at system initialization in `/cell/go.scp`.
2. The second, `er_specs` configures `evnt_rsp` and can be launched in `/cell/go.scp` or from the command line after the specifications are modified.

Refer to *Section 3 Specification File* on page 5 for related information.

2.1 event_rsp

The CyFlex process `evnt_rsp` performs the real-time processing of events. This process is started during system initialization. Refer to cyflex.com usage help for `evnt_rsp`.

2.2 er_specs

The CyFlex `er_specs` command reads the configuration information provided within a specification file to the `evnt_rsp` process and sends the information to the `evnt_rsp` applications. Refer to cyflex.com usage help for `er_specs`.

3 Specification File

3.1 Header Format

The specification file has an optional header which has the name of the event response process (`evnt_rsp`) which is to receive and process the specifications. This header should appear only once in a specification file. Specify the header as follows:

```
@REG_NAME
<registered_name>
```

where:

`name` is the name provided with the `evnt_rsp` process when it was started. Refer to the description for the `event_rsp registered_name` option in cyflex.com usage help for `evnt_rsp`.

Example:

```
@REG_NAME
plc2
```

3.2 Body Format

The format of a specifications file is a sequence of lists of trigger events, conditions, and actions. Each list has the format:

```
@INPUT_EVENT
<trigger_event>

<list of keywords for conditional tests and actions>
```

The end of the list of keywords is terminated by the next `@INPUT_EVENT` spec.

The **Keyword Description** section identifies whether a keyword is a conditional test, an action, or conditional keyword. The order of keywords within a single trigger event's specification does not matter provided the data associated with a keyword is provided after the `@INPUT_EVENT` keyword and prior to any other `@INPUT_EVENT` keywords. The examples provided are not intended to be complete specifications, but rather the portion of a specification which relates to the keyword being described. Complete listings and explanations of examples are provided in *Section 3.3 Example Specifications* on page 10.

Table 2: Keyword Descriptions

TYPE *code -- I=input, C=conditional test, P=PASS actions, F=FAIL actions, A=Always*			
Keyword	**Type**	**Description**	**Example**
`@INPUT_EVENT`	I	This is the only required entry for defining an event response. The purpose is to identify the event name or names that will cause the rest of the actions to be processed. Up to 4 events may be specified.	`@INPUT_EVENT` `abort_limit`

TYPE *code -- I=input, C=conditional test, P=PASS actions, F=FAIL actions, A=Always*			
Keyword	**Type**	**Description**	**Example**

`@IF_TRUE_LIST`	C	Provides a list of logical variables or computed expressions. All the specifications must resolve to a true condition, else the conditional state will be `FAIL`.
		If a variable doesn't exist or the expression can't be evaluated, then that item is ignored and the other valid specifications in the list are evaluated.
			`@IF_TRUE_LIST`
			`Engine_Run`
			`"RPM > 1000[rpm]"`

`@IF_FALSE_LIST`	C	Provides a list of logical variables or computed expressions. All the specifications must resolve to a false condition, else the conditional state will be `FAIL`.
		If a variable doesn't exist or the expression can't be evaluated, then that item is ignored and the other valid specifications in the list are evaluated.
			`@IF_FALSE_LIST`
			`Some_variable`
			`"A && B"`

`@PASS_PARAMETERS`	P	Specifies changes to variables if the conditional state is PASS. The value field may be a constant, variable label, or computed expression. The one-shot delay is optional.	`@PASS_PARAMETERS`
			`#variable value delay`
			`myx "1[%]+myz" 10[s]`

`@PASS_STATUS`	P	Change to the display status of variables if the conditional state is PASS.	`@PASS_STATUS`
			`#variable status delay`
			`myx BLINK`

`@PASS_OUTPUT_EVENT`	P	Set an event if the conditional state is PASS. The one-shot delay is optional.	`@PASS_OUTPUT_EVENT`
			`#event delay`
			`my_event 10[s]`

`@PASS_SCRIPT`	P	Launch a background process if the conditional state is PASS.	`@PASS_SCRIPT` `/specs/cmds/mys 14 100`
`@PASS_COMMENT`	P	Write an 80-character line to the file `/data/errors/evnt_rsp` if the conditional state is PASS.	`@PASS_COMMENT` `This is a comment line`

TYPE *code -- I=input, C=conditional test, P=PASS actions, F=FAIL actions, A=Always*			
Keyword	**Type**	**Description**	**Example**
@FAIL_PARAMETERS	F	Changes to variables if the conditional state is FAIL. The value field may be a constant, variable label, or computed expression. The one-shot delay is optional.	@FAIL_PARAMETERS #variable value delay myx "1[%]+myz" 10[s]
@FAIL_STATUS	F	Change to the display status of variables if the conditional state is FAIL.	@FAIL_STATUS #variable status delay myx BLINK
@FAIL_OUTPUT_EVENT	F	Set an event if the conditional state is FAIL. The one-shot delay is optional.	@FAIL_OUTPUT_EVENT #event delay my_event 10[s]
@FAIL_SCRIPT	F	Launch a background process if the conditional state is FAIL.	@FAIL_SCRIPT /specs/cmds/mys 14 100
@FAIL_COMMENT	F	Write an 80-character line to the file /data/errors/evnt_rsp if the conditional state is FAIL.	@FAIL_COMMENT This is a comment line
@FAIL_ERROR_CODE	F	If the conditional state is FAIL, then write an error message of this type. ERR_FILE - to /data/errors/evnt_rsp ERR_LOG - to /ram/error.log ERR_SCRN - flag the active console	@FAIL_ERROR_CODE ERR_FILE
@IF_FAILURE_DISPLAY	F	If the conditional state is FAIL, then write an indication of the conditional tests which caused the FAIL state to this string variable.	@IF_FAILURE_DISPLAY NOTIFY
@EMAIL	A	Sends an email, text message, or pager message.	@EMAIL #address message subject joe@cummins.com 'problem'

TYPE *code -- I=input, C=conditional test, P=PASS actions, F=FAIL actions, A=Always*			
Keyword	**Type**	**Description**	**Example**
@LOOKUP	A	Assign a value to a target variable as a function of the value of a numeric variable value or expression.	@LOOKUP #TargetVariable CartModeDesc #Input Expression CartModeInt #ExpressionUnits TargetUnit None None #DefaultTargetValue “Value Not Mapped” #ExpressionValue Tolerance TargetValue 1 0.01 “OFF” 2 0.01 “IDLE” 3 0.01 “FLUSH” 4 0.01 “PURGE”
@PASS_LOOKUP	P	If the conditional state is PASS, then assign a value to a target variable as a function of the value of a numeric variable value or expression.	@PASS_LOOKUP #TargetVariable CartModeDesc #Input Expression CartModeInt #ExpressionUnits TargetUnit None None #DefaultTargetValue “Value Not Mapped” #ExpressionValue Tolerance TargetValue 1 0.01 “OFF” 2 0.01 “IDLE” 3 0.01 “FLUSH” 4 0.01 “PURGE”

TYPE *code -- I=input, C=conditional test, P=PASS actions, F=FAIL actions, A=Always*			
Keyword	**Type**	**Description**	**Example**
@FAIL_LOOKUP	F	If the conditional state is FAIL, then assign a value to a target variable as a function of the value of a numeric variable value or expression.	@FAIL_LOOKUP #TargetVariable CartModeDesc #Input Expression CartModeInt #ExpressionUnits TargetUnit None None #DefaultTargetValue “Value Not Mapped” #ExpressionValue Tolerance TargetValue 1 0.01 “OFF” 2 0.01 “IDLE” 3 0.01 “FLUSH” 4 0.01 “PURGE”
@ELOG	A	Create an automatic entry into the electronic logbook. The message string can be a literal string enclosed in single quotes, the label of a CyFlex string variable, or point to a text file whose contents will be used as the message. A single logbook entry per INPUT_EVENT is allowed. Also, 2 entries cannot be made simultaneously, so creating multiple INPUT_EVENT instances with the same trigger event will not result in multiple logbook entries.	@ELOG #WWID message to_mcparts li123 ‘automated message’ FALSE
@FAIL_ELOG	A	If the conditional state is FAIL, then create an automatic entry into the electronic logbook.	@FAIL_ELOG #WWID message to_mcparts li123 ‘fault message’ TRUE

TYPE *code -- I=input, C=conditional test, P=PASS actions, F=FAIL actions, A=Always*			
Keyword	**Type**	**Description**	**Example**
@PASS_ELOG	A	If the conditional state is PASS, then create an automatic entry into the electronic logbook.	@PASS_ELOG #WWID message to_mcparts li123 'passed test message' TRUE

3.3 Example Specifications

```
##############################################################################
# Classification of keywords:
#  input or trigger - any one of the events will trigger processing of the
#                     the spec
#     keyword                    maximum number perf specification
#     @INPUT_EVENT               ( 4 per spec)

#  conditionals - the states determine whether PASS or FAIL "actions"
#                 are enabled
#     keyword                    maximum number perf specification
#     @IF_TRUE_LIST              (32 per spec)
#     @IF_FALSE_LIST             (32 per spec)

#  PASS_ actions   - the implied operations are performed if all of
#                    the conditional flags ARE satisfied
#     keyword                    maximum number perf specification
#     @PASS_PARAMETERS           (64 per spec)
#     @PASS_STATUS               (16 per spec)
#     @PASS_OUTPUT_EVENT         ( 8 per spec)
#     @PASS_SCRIPT               ( 1 per spec)
#     @PASS_COMMENT              ( 1 per spec)

#  FAIL_ actions   - the implied operations are performed if any of the
#                    conditional flags are NOT satisfied
#     keyword                    maximum number perf specification
#     @FAIL_PARAMETERS           (64 per spec)
#     @FAIL_STATUS               (16 per spec)
#     @FAIL_OUTPUT_EVENT         ( 8 per spec)
#     @FAIL_SCRIPT               ( 1 per spec)
#     @FAIL_COMMENT              ( 1 per spec)
#     @FAIL_ERROR_CODE           ( 1 per spec)
#     @IF_FAILURE_DISPLAY        ( 1 per spec)

##############################################################################
#                      keyword formats & examples
#
#  @INPUT_EVENT
#     #event_name (up to 4)
#     push_button
#     button_cmds

#  @IF_TRUE_LIST
```

```
#     #variable label or expression (up to 32)
#     Engine_Run
#     " ctl_spd > 1000[rpm]  &&   oil_rifle_p < 10[psi] "
#
#  @IF_FALSE_LIST
#     #variable label or expression (up to 32)
#     starter
#     flame_detect
#     coolnt_level

#  @PASS_PARAMETERS    or     @FAIL_PARAMETERS
#     #label            value             delay(optional)
#     key_switch        OFF
#     ecm_variable      'USC_FUEL'
#     ecm_value         100[none]
#     beep              ON
#     beep              OFF               5[sec]
#     NOTIFY            "taking action now"
#     myvar             "if( ctl_spd > 0[rpm] ) then ctl_spd else perf_spd"

#  @PASS_OUTPUT_EVENT   or    @FAIL_OUTPUT_EVENT
#     #event_name       delay(optional)
#     set_tvo_sig
#     report_data       1[min]

#  @PASS_STATUS         or    @FAIL_STATUS
#     #label            status_value            delay(optional)
#     key_switch        BLINK
#     key_switch        NORMAL                  10[sec]
#     beep              BLINK_RED
#     beep              NORMAL                   5[sec]

#  @PASS_SCRIPT         or     @FAIL_SCRIPT
#     #command
#     /bin/cleanup "/data/stuff/*"  1 100

#  @PASS_COMMENT        or     @FAIL_COMMENT
#     #comment line (up to 80 characters)
#     use this line as an error message into /data/errors/evnt_rsp

#  @FAIL_ERROR_CODE
#     #code
#     ERR_SCRN

#  @IF_FAILURE_DISPLAY
#     #label of string variable
#     ER_expl

##############################################################################
# The following implement a variable frequency pulse-width modulation output
# where "egr_interval" defines the frequency (inverted) and "egr_duration"
# defines the output on time (in milliseconds)

# attach to a 20 millisecond timer
@INPUT_EVENT
tmr-20
```

```
# see if the interval has expired (this is inverse of the base frequency)
@IF_TRUE_LIST
"pulse_interval >= egr_interval && egr_duration > 0[ms] "

# the FAIL actions are for when the interval has NOT expired - just
# keep incrementing the interval counter by the value of the timer period
@FAIL_PARAMETERS
pulse_interval    "pulse_interval + 20[ms]"

# the PASS actions are for when the interval has expired - reset the
# interval counter, turn the output ON, and reset the output duration
# counter
@PASS_PARAMETERS
pulse_interval    0[ms]
egr_out           ON
pulse_duration    0[none]

@INPUT_EVENT
tmr-20

# see if the output duration has exceeded the desired pulse width
@IF_TRUE_LIST
"pulse_duration >= egr_duration"

# the pulse width has exceeded specification - turn it OFF
@PASS_PARAMETERS
egr_out        OFF

# not expired - increment the pulse curation counter
@FAIL_PARAMETERS
pulse_duration    "pulse_duration + 20[ms]"

#attach to a 1 second timer
@INPUT_EVENT
tmr-1000

#assign the string value of CartModeDesc based on the integer value of
#CartModeInt
@PASS_LOOKUP
#TargetVariable
CartModeDesc

#Input Expression
CartModeInt

#ExpressionUnits TargetUnits
None             None

#DefaultTargetValue
“Value Not Mapped”

#ExpressionValue Tolerance TargetValue
1                0.01     “OFF”
2                0.01     “IDLE”
3                0.01     “FLUSH”
```

```
4                0.01     “PURGE”
@ELOG
# wwid          message                        send_to_mcparts
li123    ‘automated message – out of fuel’   TRUE
```