Test Manager Procedure File Keyword Organization

Version 4

February 13, 2024

Developed by Transportation Laboratories

Version History

Version	Date	Revision Description
1	1/25/2016	Initial publication
2	8/23/2018	Add SGS branding
3	4/2/2020	Retrofit to new template
4	2/13/2024	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 Keyword Organization Overview* on page 1.

  The clickable cross-references in the preceding example are *1*, *Keyword Organization 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 Keyword Organization Overview

The organization of keywords in a test procedure file has a prescribed format that consists of the following components:

- `Start_mode`
- Global procedure keywords
- Test Mode specifications

1.1 Start_mode

The file must always begin with a single line that is the initial mode number that will be executed when the procedure is started or called as a sub-procedure. This is referred to as the "start_mode".

1.2 Global Procedure Keywords

Eight (8) option keywords follow the `start_mode` that define actions that the Test Manager will execute and associated with this procedure and in some cases, sub-procedures These global keywords are different than the keywords which are used in the "instance defintion file", which has a somewhat similar function, but apply to the instance of the Test Manager regardless of which procedures are executed.

1.3 Test Mode Specifications

Following the optional "global procedure keywords" section the next line must be an `@MODE` keyword. The rest of the file consists of anywhere from 1 to 999 blocks defining the various test operations Each of the blocks begins with the `@MODE` keyword and is terminated by the next `@MODE` keyword. Note that mode numbers are just tags and do not have to be in any particular order in the file on in execution sequence. However, it is usually easier to understand a procedure if modes appear in sequence in the file.

2 Organized Keywords

2.1 Instance Name

Each test procedure file can be designated as belonging to a particular instance of the Test Manager. Each instance (copy) of the Test Manager that is running in a system must have a unique name. This is referred to as the “instance name”. The instance name is defined in the “instance definition file” which is unique for each Test Manager that is launched. By using the `@INSTANCE` keyword in a test procedure file, the procedure will be tagged as belonging to that instance and will not be allowed to be executed by any other instance. Every root procedure should include this keyword, but it is not mandatory.

```
@INSTANCE
   #instance name
   test
```

Refer to *Instance Definition File* in the *Test Manager User Guide* for related information.

2.2 Creating Events

Create events of zero length in the global section using this keyword. This would normally be done if these events are used only when this test procedure is being run. When a new procedure is started using the `nt` command, these events will be destroyed. Refer to *CyFlex Events* for a description of how to create a permanent event.

```
@CREATE_EVENT
     # list of event names – up to 24
     My_event1
     My_event2
```

2.3 Creating Variables

Basic `REAL`, `INTEGER`, `LOGICAL`, and `STRING` variables can be created in the global keywords section. This would normally be done if these variables are used only when this test procedure is being run. When a new procedure is started using the `nt` command, these variables will be destroyed.

```
@CREATE_VAR
# list of  – up to 128 variables
#label  type    units   initial_value  display_resolution
Mynewx  REAL    psi     100[in_hg]   4
Count     INT     none    0[none]
Mystate LOGI    none    “if A then B else C”
Mystring  STRING  -     ‘up to 80 char’
```

Note:

> the initial value may be set by the result of a computed expression (double quoted). This computed expression is evaluated only once to set the initial value. See *Section 2.4 Creating Expressions* on page 3 to assign a computed expression that will continually update the value of a variable.

Refer to *Creating User Computations and User Variables* to create permanent variables.

2.4 Creating Expressions

Basic REAL, INTEGER, LOGICAL, and STRING variables can be created in the global keywords section. This would normally be done if these variables are used only when this test procedure is being run. When a new procedure is started using the nt command, these variables will be destroyed

The target variable (label) will be created if it does not already exist.

```
@CREATE_EXPRESSION
# list of  – up to 128 variables
#label    type    units   event/timer  expression
Mynewx    REAL    psi       SLO   “if A then 10[psi] else 0[psi]”
Mystring  STRING-          SLO   “ ‘test’ + count “
```

2.5 Creating Transition Events

This provides the ability to attach specifications for transition events to any LOGICAL variable created by this instance of the Test Manager. This must be entered in the file after the creation of the variable by the @CREATE_VAR keyword. Both the variables and events are destroyed when a new procedure is started with the nt command.

```
@CREATE_TRANSITION EVENT
# list of up to 128 variables
# logical_variable  true_event_name   false_event_name
Mystate      state_on       state_off
```

2.6 Global Events

The @GLOBAL_EVENTS specifications define paths that will be taken in response to events while this procedure is being executed regardless of which test mode is currently active. The event name may be any event in the system.

If neither the next_mode nor test_procedure are specified, then the current mode is terminated and the normal default_next_mode path is taken.

If the next_mode is specified, but the test_procedure is not, then the test will jump to that mode in this procedure.

If the test_procedure is specified, but the next_mode is not, then the test will jump to that sub-procedure. If the sub-procedure exits via a RETURN, then the current procedure will begin executing at the default_next_mode specified for the test mode that was interrupted.

If both next_mode and test_procedure are specified, the next_mode will be ignored, unless the test_procedure file does not exist or cannot be opened. In that case the next_mode path in this procedure would be taken.

```
@GLOBAL_EVENTS
#event_name       next_mode      test_procedure
emergency      -         /specs/gp/gp_emergency
abort_limit       -         /specs/gp/gp_shutdown
```

2.7 Registered Events

@REGISTERED_EVENTS are events that are sent by the following CyFlex commands. These two events are currently the only registered events that allow the user to control the execution path.

```
#command        event_name
stop         stop_test
idle         idle_mode
```

The events sent by these commands are specific message events that contain the name of the instance of the Test Manager that they are intended to signal. For example, if the instance of gp_test named test_map is to be stopped, type the command:

```
stop test_map
```

Each instance of the Test Manager in the system will receive the stop_test event if they have it listed as a registered event in the currently executing procedure, but all those whose instance name is not test_map will ignore the message.

This assumes that the stop_test event is specified as a @REGISTERED_EVENTS and that the path specified is appropriate to stop the test procedure.

The format and handling of the paths for @REGISTERED_EVENTS is the same as for @GLOBAL_EVENTS, differing in that the only those events listed above can be specified, since they are the only messages that will contain an instance name.

```
@REGISTERED_EVENTS
#event_name       next_mode      test_procedure
stop_test      -         /specs/gp/gp_shutdown
idle_mode      99        -
```

2.8 Spawning a Co-Process

If an application or “co-process” is needed while a particular test procedure is running, but only while that procedure is running, it can be launched using this keyword specification. The application will be automatically killed when a new test procedure is started with the nt command.

```
@SPAWN_CO_PROCESS
#list up to 16 process with appropriate arguments – do not
enclose in quotes
   /cyflex/bin/floger /specs/flogging +W
   /cyflex/bin/floger /specs/flogon
```