Manufacturing Test Executive Theory of Operation

Document Number M000087-01, Revision C

(Applies to Test Executive V2.1.X.X)

By Edward Kaetz, 1/8/2019

1 Introduction

1.1 Overview

The Manufacturing Test Engineering (MTE) Test Executive (TE) is a generic utility that is intended to be deployed on all test stations. It provides a common user interface to the test operator and it provides interfaces to other manufacturing support systems.

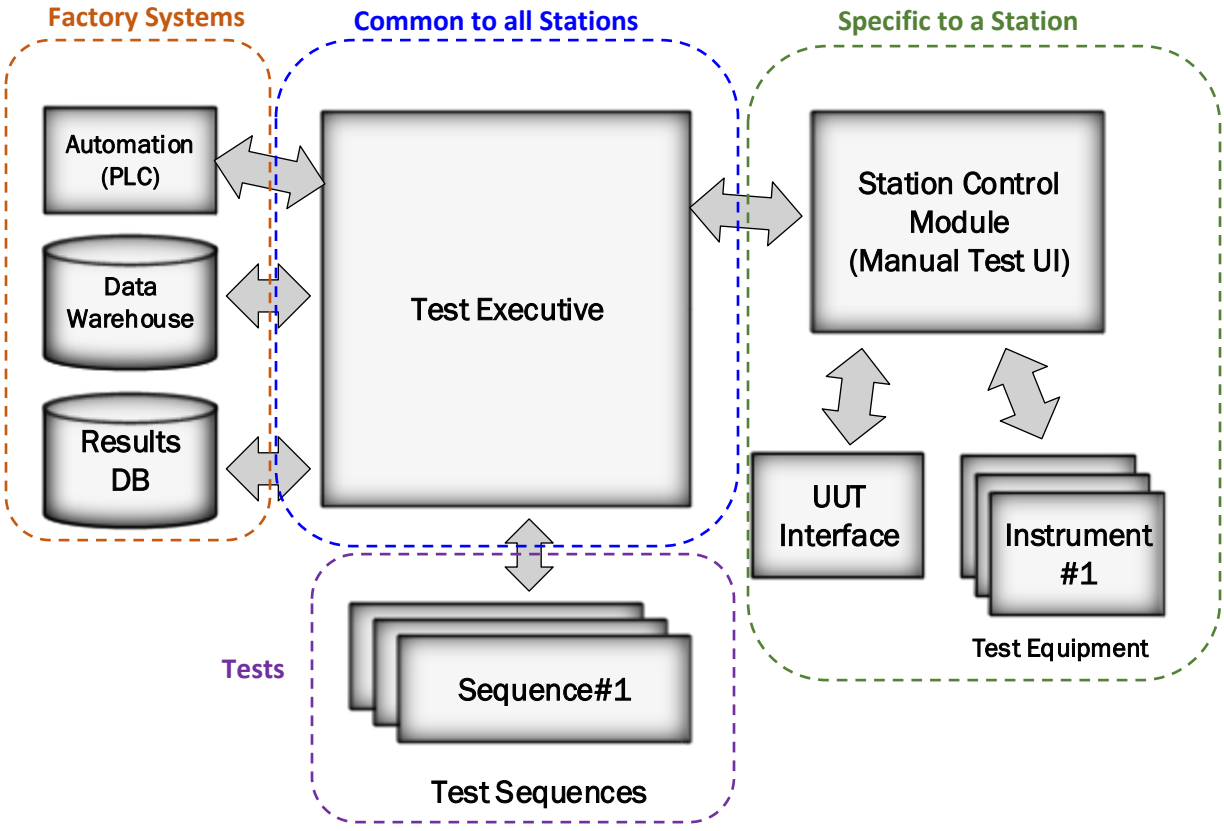


Figure 1 - Test System SW Architecture Diagram

Each test station will have a Station Control Module which is software that controls all equipment on the test station and the unit under test (UUT). This module can be used to manually perform tests and to perform verification and debugging of test station hardware and software. This module will have an interface where it can be remotely controlled by the Test Executive to perform test sequences.

The test executive has controls to select and run test sequences. These sequences are separate files that are released separately from the test executive. The sequences will call functions in the station control module that will perform test setup and measurements. Test results are collected and displayed by the test executive. When the sequence ends, a test report is sent to the manufacturing test result databases.

The test executive will interface to the shop floor control (SFC) system. It will report when a UUT test is started and completed. This is used for routing and tracking of UUTs.

The test executive will interface to the factory calibration system to obtain calibration status of equipment. If an instrument is out of calibration the operator will be notified.

The test executive is configured to take the identity of the test station it is deployed on. The configuration provides:

- Test Station Name - this is displayed at the title of the Test Executive user interface.
- Location of the Station Control Module – this is run in the background and managed by the test executive.

1.2 Purpose

The purpose of this document is to provide instructions for the installation, configuration, and operation of the MTE Test Executive.

1.3 Scope

This document is intended to be used by the test engineer to install and operate the test executive. This document provides too many details for the general operator. The test engineer should us this information to generate work instructions and to train operators.

1.4 Reference Documents

TBD

1.5 Acronyms

Acronym	Description
TBD	To Be Determined
UI	User Interface
TE	Test Executive
UUT	Unit Under Test
SFC	Shop Floor Control
DB	Database
MTE	Manufacturing Test Engineering

1.6 Table of Contents

1.7 Table of Figures

2 User Interface

The Test Executive user interface is shown in Figure 2. It consists of sections to provide:

- Test System Identification
- Test Control
- Test Status
- Test Results
- Test Equipment List
- Test System Health
- Test Metrics
- Configuration and Maintenance

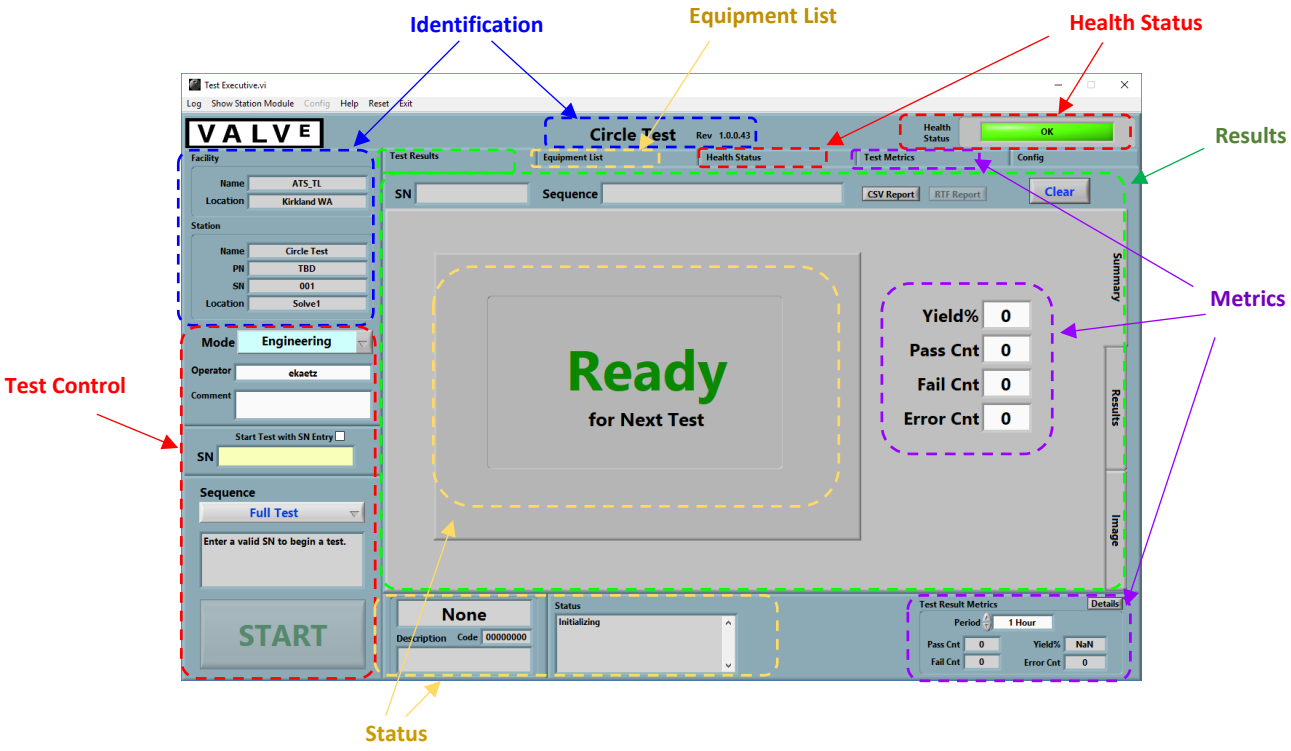


Figure 2 – User Interface, Overview

2.1 Test System Identification

The UI displays facility and station information in the top right corner (see Figure 2). This information is obtained from configuration files. See section 5 for configuration details.

The title of the application is displayed in the top center of the UI. This is the test station name and is obtained from configuration provided by the installer.

2.2 Test Control

The UI provides test control along the right side of the UI (see Figure 2).

2.2.1 Prior to Running a Test

Set the Mode and select a sequence.

Enter comments describing the test.

Enter the Operator and UUT SN.

- Mode (Combo Box)
  - The operator can select a mode
    - NOTE: Mode selection will be password protected in the future.
  - The Mode name provides an attribute on the test results to be used to filter the results based on mode.
  - The Mode has configured options that change the behavior of the TE. See section 3 for details about Mode Options. Certain features are only available in specific modes. Production mode is the most restrictive mode and only enables features necessary for production testing activities.
- Operator Name (Text Box)
  - The default name is the Windows user. The operator can over write this field with a different name.
  - A test cannot be run unless the Operator field is populated.
- Comment (Text Box)
  - This field is editable by the operator.
  - It can be used to enter key words about the test setup that can be used to filter the test results.
- Sequence (Combo Box)
  - When the TE launches it defaults it selects the configured default test sequence.
  - Based on the option settings for the mode, the operator may or may not be able to select a different sequence.
- SN (Text Box)
  - The operator must enter a serial number for the UUT. This is typically done with a barcode scanner.
  - A test cannot be run unless the SN field is populated.
  - NOTE If the 'Start with SN Entry' check box is selected, the test will start once the SN is entered.

2.2.2 Launching a Test

If the 'Start with SN Entry' check box is selected, the selected sequence will begin once the UUT SN is entered.

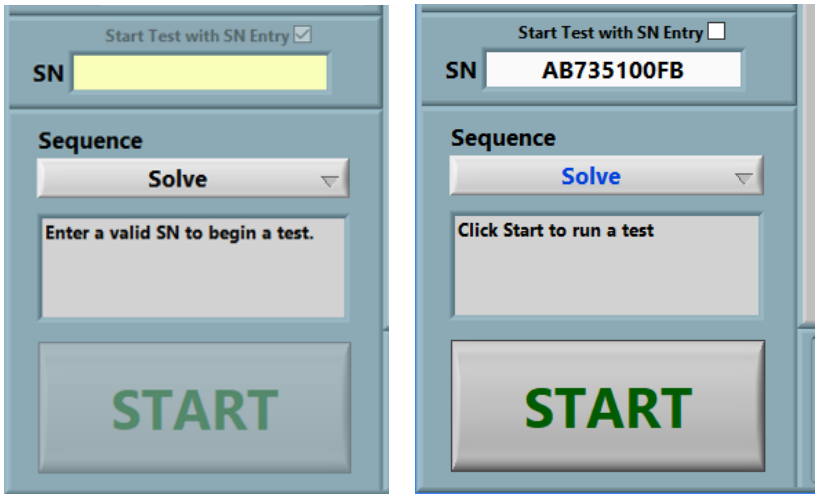


Figure 3 - Start Button is Disabled Until a SN is Entered

If the 'Start with SN Entry' check box is not selected, the START button will be enabled if the Operator and SN fields are populated. When START is pressed, the selected sequence will begin.

2.2.3 While a Test is Running

Test Status is displayed in the lower left corner and on the Result Summary page. If the mode is Production (or other more restrictive mode) the detailed results are not visible. Only a result summary is visible as shown in Figure 4.

In Technician or other less restrictive modes the detailed results page can be displayed as shown in Figure 5.

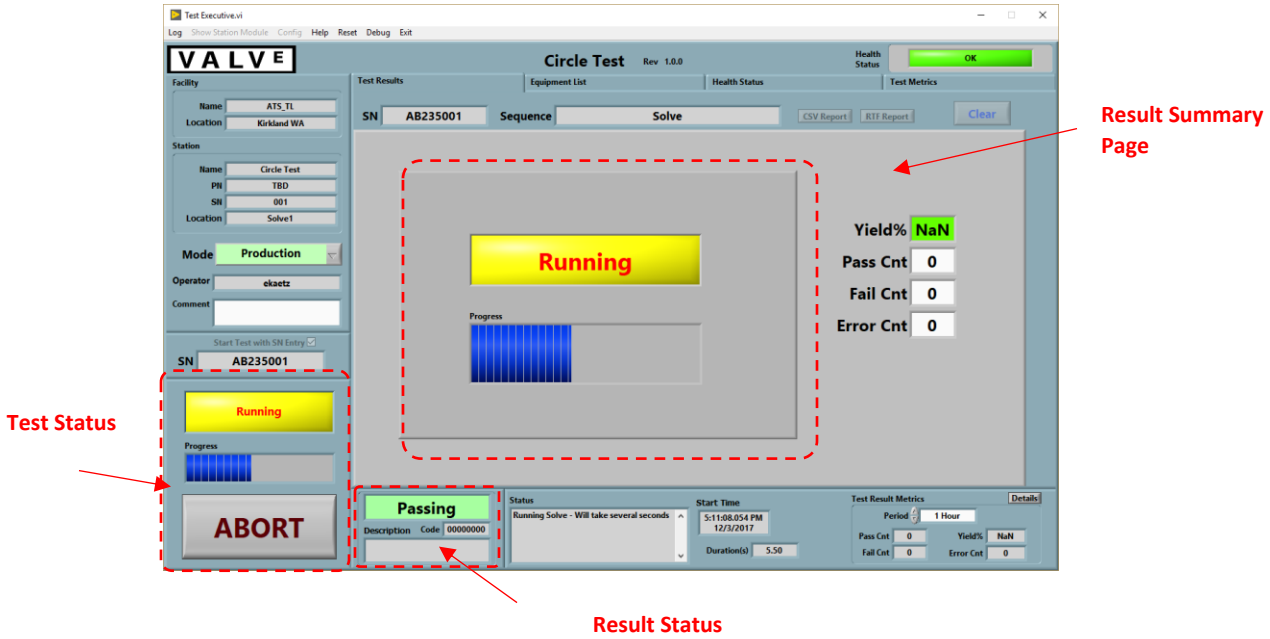
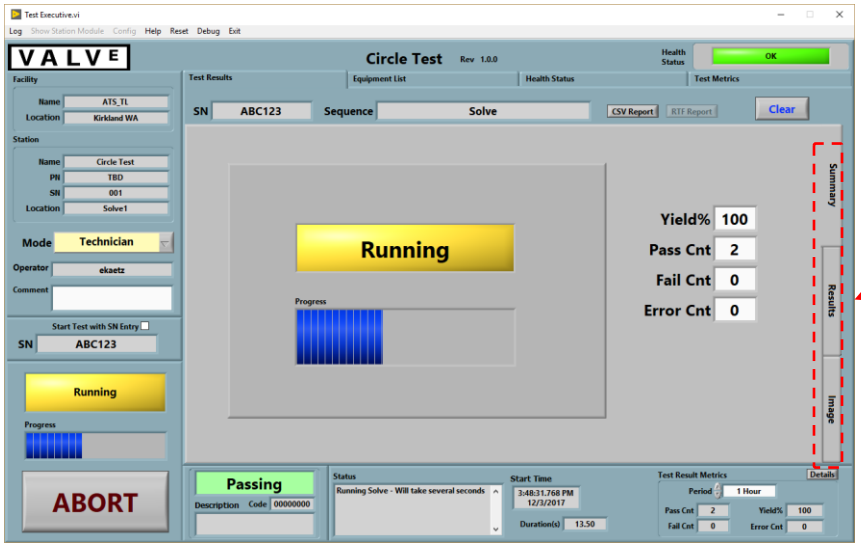


Figure 4 - Test Status – Detailed Results Hidden



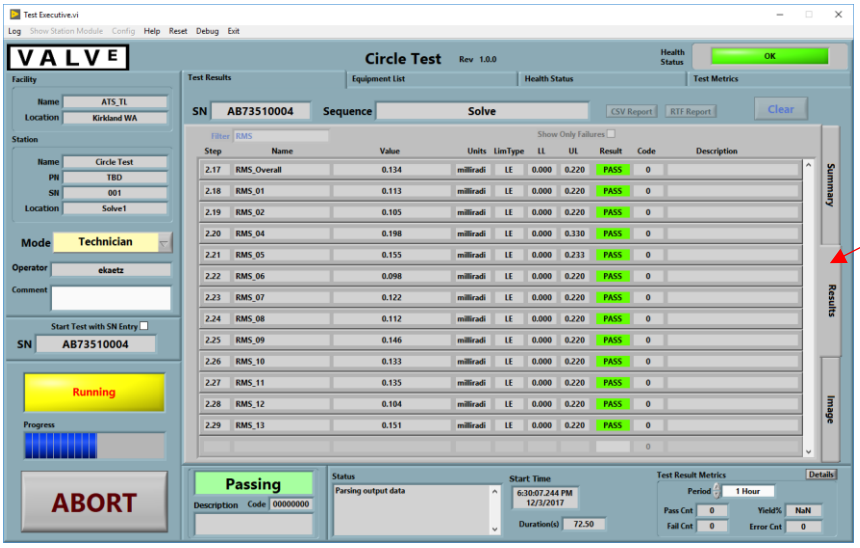


Figure 5 - Test Status – Detailed Results Displayed

2.2.4 Operator Image Prompt

The TE can prompt an operator to view an image and select YES or NO buttons to provide PASS/FAIL input. A function in the test sequence loads the image and initiates the prompt action.

When the operator is prompted to view and image and provide input:

- The Result section switched to the Image page (as shown in Figure 6).
- Status text box at the bottom center of the UI blinks yellow and indicates the TE is waiting for operator input.

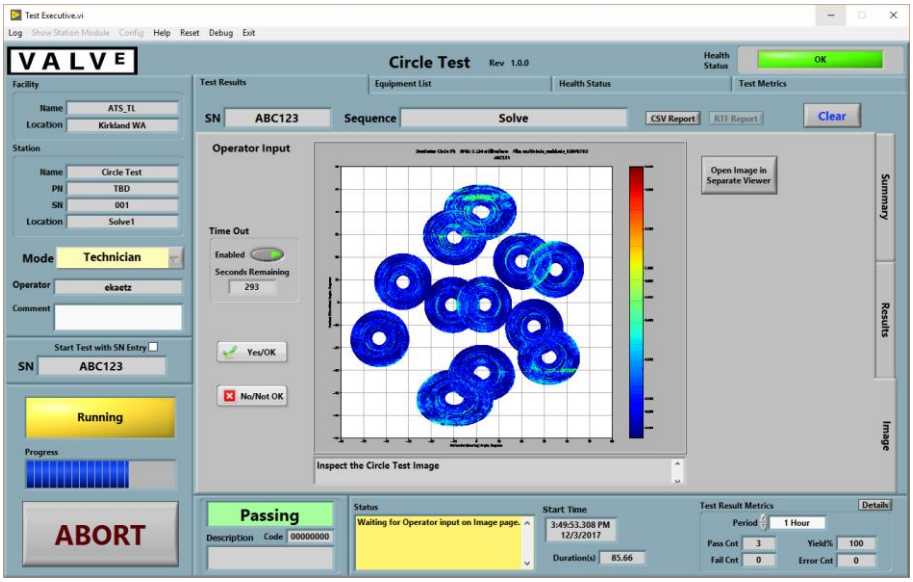


Figure 6 – Image Prompt Page

On the image prompt page:

- An image and a prompt are displayed.
- There are Yes and No buttons for the operator to select PASS/FAIL.
- There is a time out indicator that is counting down.
    - If the operator does not provide input within 5 minutes (300 seconds) the test will abort due to no response.
    - The time out can be disabled by turning the Enabled switch off in the Time Out control section.
- There is a button “Open Image in Separate Viewer”
    - When this button is clicked, the image file is opened in the default image viewer for windows (see Figure 7). This will allow the operator to view the image in a larger window.

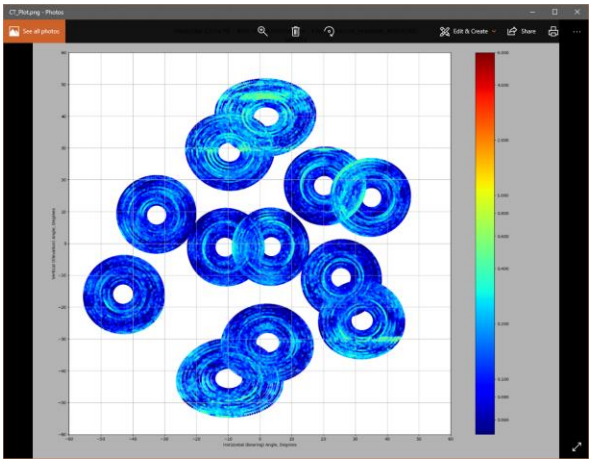


Figure 7 – Open Image in Separate Viewer

2.2.5 Operator Data Prompt

The TE can prompt an operator to view an image and select YES or NO buttons to provide PASS/FAIL input. A function in the test sequence loads the image and initiates the prompt action.

When the operator is prompted to view and image and provide input:

- The Result section switched to the Image page (as shown in Figure 6).
- Status text box at the bottom center of the UI blinks yellow and indicates the TE is waiting for operator input.

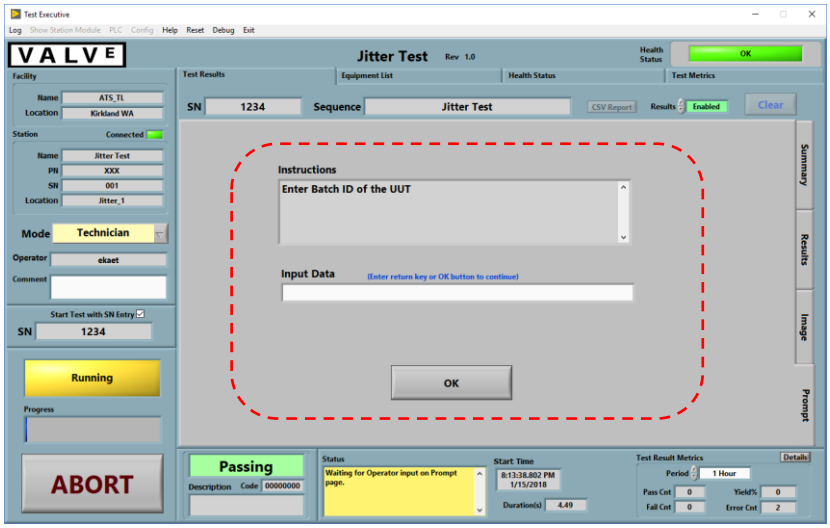


Figure 8 – Data Prompt Page

On the data prompt page:

- The operator is provided instructions to explain what data is needed.
- When the operator modified the input data field, or the OK button is pressed the test will resume.

2.2.6 Once a Test is Completed

When a test is completed, the overall results are displayed in the Result Summary Page and in the status section at the bottom of the main UI as shown in Figure 9. The overall result will indicate PASS, FAIL or ERROR.

- PASS or FAIL indicate the UUT has passed or failed.
- ERROR indicates some unexpected issue occurred during the test. When an ERROR occurs, the suspected cause is an issue with the test system not the UUT. The test system should be investigated prior to troubleshooting the UUT.

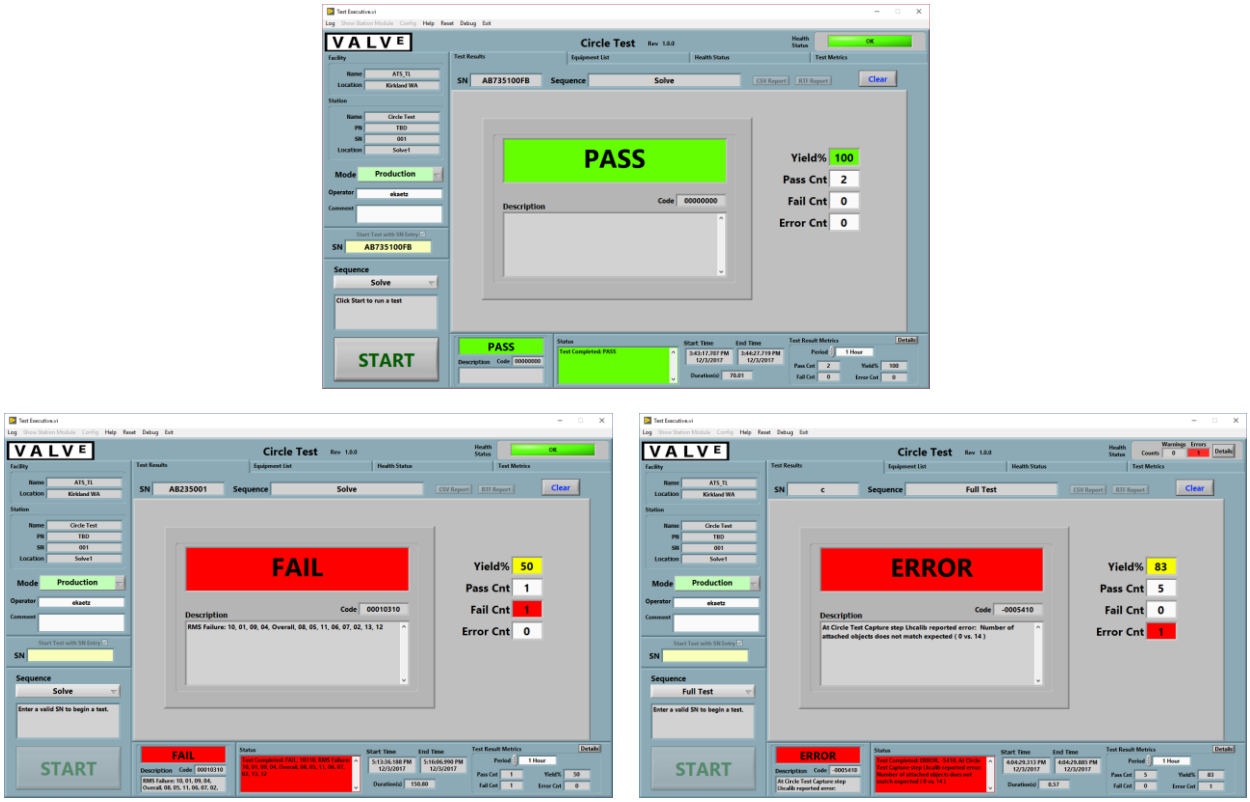


Figure 9 – Test Overall Result

2.3 Equipment List

Tet TE receives an updated the equipment list from the test station module when it launches an automated test. The station module may auto generate its equipment list, or it may require the operator to enter data to produce the equipment list. See the user manual for the specific station to get details on how to manage the equipment list for a specific test station.

The TE displays the equipment list on the Equipment List tab as shown in Figure 10.

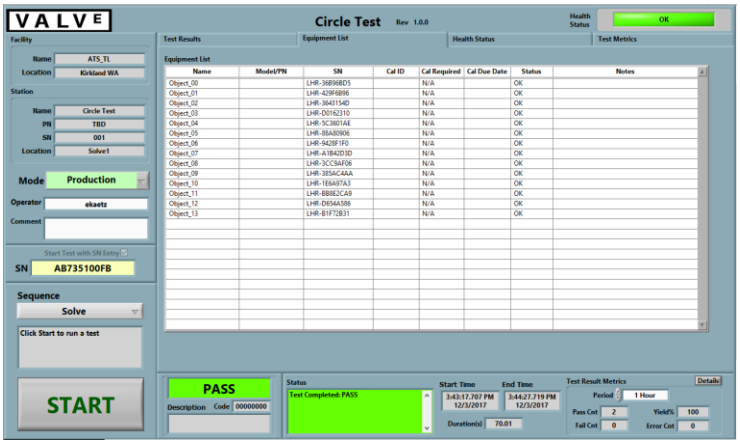


Figure 10 – Equipment List

2.4 Test Metrics

The TE keeps track of the test result history. This history is limited to 10,000 records. Once its max size is reached, any new result over writes the oldest result.

The test metrics is displayed on the Result Summary page, on the lower right corner of the UI, and in the Test Metrics tab (see Figure 2).

The time range of the test metrics can be selected in the lower right corner of the UI. The available ranges are: 1 hour, 4 hours, 1 day, 1 week.

The Test Metrics tab contains a list of the test results history (see Figure 11).

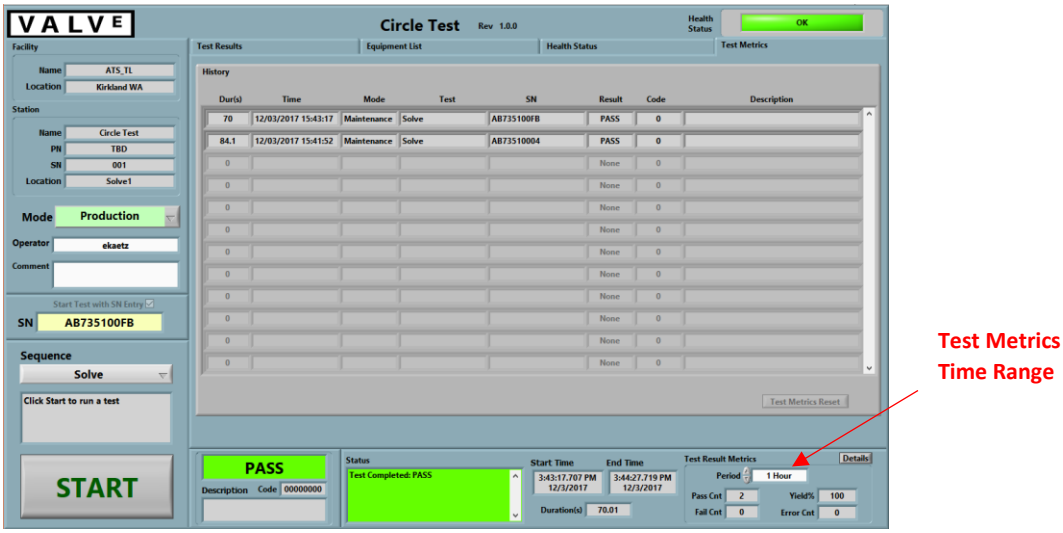


Figure 11 – Test Metrics Tab

2.5 Health Status

The TE health status is indicated on the top right corner of the UI. The TE keeps track of warnings and of errors that have occurred. If there are no warnings or errors, the health status is indicated as 'OK' with a green light.

If warnings or errors have occurred then the OK green light changes to a 2 counters, one is yellow for warnings and one is red for errors.

A list of errors and warnings are displayed o the Heath Status page (see Figure 12). On the Health Status tab, there are controls to clear the warnings and errors. These controls are enabled or disabled based on the selected Mode and the options for that mode.

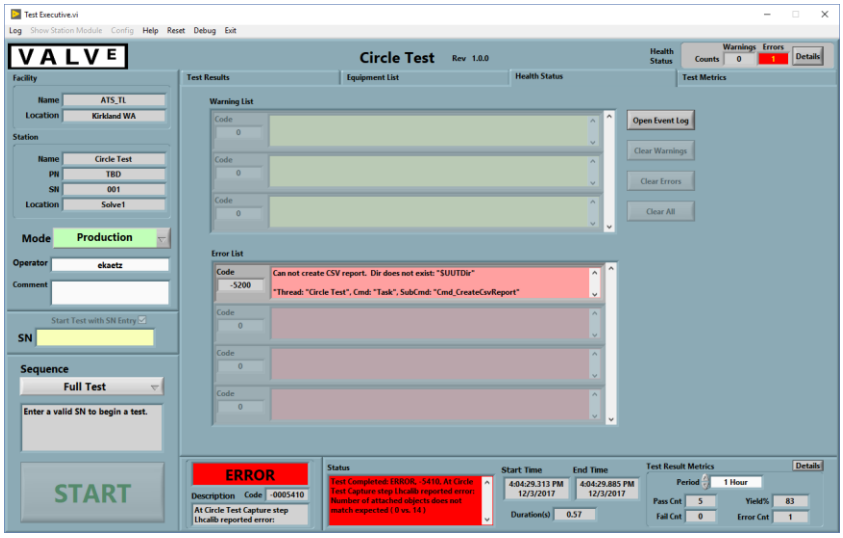


Figure 12 – Test Health Tab

3 Mode and User Interface Options

The TE has a Mode selector. Each mode has a name, a color, and a definition of options that setup the behavior of the TE for that mode.

The Mode name is recorded with each test result. This allows the results to be filtered based on the mode.

The TE has provisions to change the behavior and to hide/show UI features based on configured options. These options are defined for each mode.

The modes are configured using a Mode Configuration Utility dialog that is described in section 5.3.

Each time the application is launched or reset, the Mode configuration is read.

The mode configuration specifies a default mode. When the application is launched or reset, it selects the default mode.

4 Installation

This section describes installation of the Test executive. The station application must also be installed. See setup procedures for the individual stations for details.

4.1 PC Requirements

The TE can be installed on a desktop or laptop computer. The following minimum requirements aer for the TE application. Since TE runs on the same computer as the test station software, the actual PC requirements may be more in order to support the test station functionality. See the test station user manual for details of the PC requirements.

Table 1 – Minimum Computer Requirements

PC Component/Feature	Requirements
Processor	Pentium 4 G1 (or equivalent) or later (64-bit)
RAM	4 GB
Screen Resolution	1920:1080 pixels
Operating System	Windows 10 64 bit
Hard Drive	256 GB

4.2 Installation

Test Executive has been developed using National Instruments LabVIEW. It requires the LabVIEW run time engine (RTE). There will be a single installer with a setup utility that installs both the application and the RTE.

Currently the TE is installed by the same installer that installs the test station SW. See installation instruction sin the user manual for the target test station.

The first time the application is installed on a new PC the installation will take several minutes and will require a reboot afterwards. This is due to the installation to the RTE.

Subsequent installations for updates will take several seconds and will not require the PC to be rebooted.

To install this application, run the setup.exe file in the installation folder.

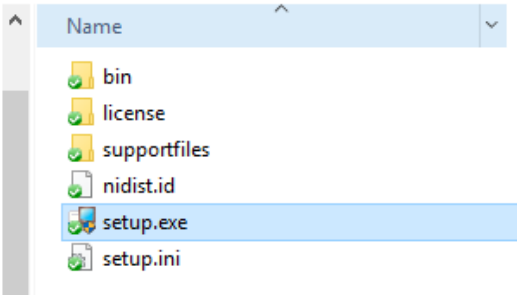


Figure 13 – Setup.exe Install File

The installer will open an installer window for Windows applications. Follow the prompts on the installer windows. A license agreement will need to be accepted.

When the installation is completed, if this was the first time the Test Executive was installed, the LabVIEW runtime engine was also installed, a reboot will be required.

4.3 Software Dependencies

All software components needed by the TE are deployed by the test station installer.

It is recommended to also install Adobe Reader so the user manuals can be viewed on the test station.

TE looks for configuration files when it runs. See section 5. The configuration files must be in place or TE will indicate an error (they cannot be found).

5 Configuration

5.1 Test Executive Configuration

The Test Executive (TE) is a common test system application. It assumes the identity of the station that it is running on.

When the TE launches, it will read the test station identity form the file:

C:\MTE\Test Executive\Config\Test Executive Config.ini

This file contains a section named 'Station' which contains the following items:

- StationName – This is the name of the test station the TE will be managing. The working folders for this station name will be used (see section **Error! Reference source not found.**).
- StationIPPort – This is the TCP-IP port used to communicate to the station application.
- StationExePath – This is the path to the executable file of the station application.

Example Config File Contents:

```
[Station]
StationName = "Base Station Final Functional Test"
StationIPPort = 8500
StationExePath = "c:\program files (x86)\valve\base station final
functional test\base station final functional test.exe"
```

This configuration file can be setup using a dialog that opens from the TE Menu:

Config -> Test Executive

The dialog window has the following controls:

- Company – This indicates which subfolder in the Windows Program Files director to search for applications.
- Type – This indicates which Windows Program Files director to use:
    - 64 Bit: C:\Program Files
    - 32 Bit: C:\Program Files (x86)
- Available Applications – this is a dropdown list that lists the applications in the selected Windows Program Files folder. When this control is changed, the Station Application path and the Station Folder are automatically populated for the selected application.
- Station Application – This is the path to the station application. It can be manually entered or auto populated by selecting the Available Application control.
- Station Folder – This is the path to the station working folder. It can be manually entered or auto populated by selecting the Available Application control. When this value is change a Boolean LED next to the control will be GREEN or RED to indicate of this folder already exists or not.
- Station App Listening Port – This is the port used for TCP-IP communication between the Station application and the TE. Both the Station application and the TE obtain the port number from this file so this port only needs to be configured in one place for both applications. This port must be set to an available port on the PC. It is not recommended to change this from the default value of 8502.

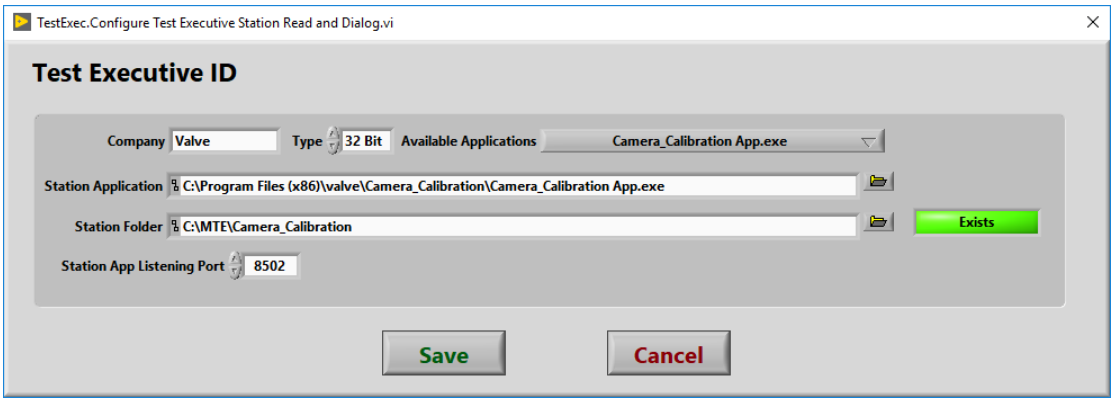


Figure 14 – Test Executive Configuration Dialog

5.2 Station Configuration

Each time the TE application is launched or reset, it reads the configuration files.

The station configuration is a combination of info contained in the Local Station file plus optional additional configuration files that may be located on network locations. When the Test Executive launches it reads the Local Station INI file:

C:\MTE\[App Name]\Config\Local Station.ini

That file may point to additional files using the AdditionalConfig section. Each file in the AdditionalConfig section will be read. If any of those files have AdditionalConfig sections those files will be read also. If any configuration items overlap, the information will be overwritten by the last configuration that is read. The final configuration will be written to a temporary file named:

C:\ProgramData\[App Name]\Working Config.ini

This file may be inspected to verify the distributed configuration is correctly being read.

NOTE: For backward compatibility, the section named “Station” will also be treated as an AdditionalConfig section which points to other config files to be merged into the working configuration.

The following sections in this document describe the configuration INI file sections.

The ability to have configuration distributed makes it easier to manage a large production environment.

An example of how the configuration can be distributed is:

- Local Station INI file
    - Contains this test station PN and SN and Factory Location Info
    - AdditionalConfig section points to Station Type Configuration
- Station Type Configuration
    - This file contains items that are common to all test stations of this type
    - AdditionalConfig section points to Product Line
- to Product Line
    - This file contains items that are common to all test stations of this product line

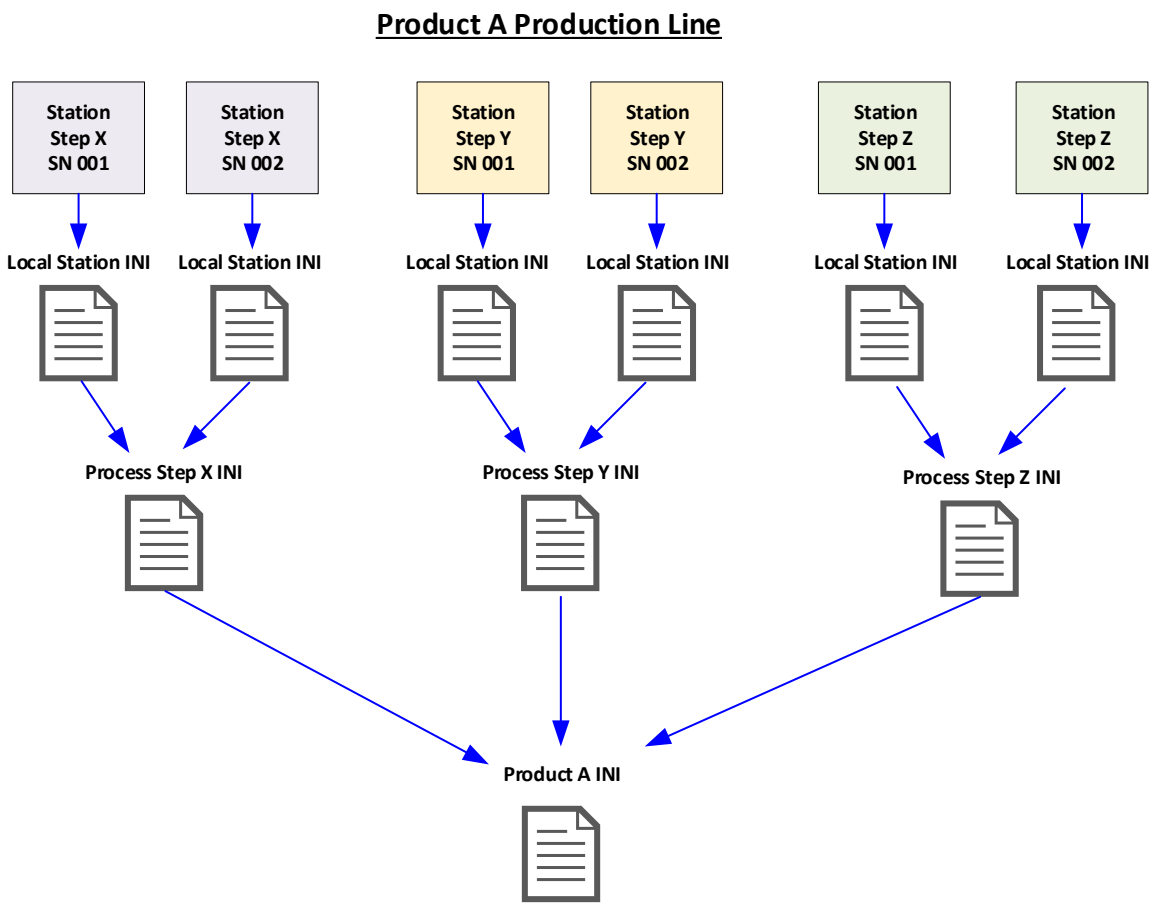


Figure 15 – Distributed Configuration

5.2.1 ID Section

This section contains ID info of the test station:

```
[ID]
Name = "Station Name"
PN = "ABC123"
SN= "001"
Rev = "-"
```

This info is displayed on the top right corner of the UI.

The ID and Factory Location configuration sections can be setup using a dialog that opens from the TE Menu:

Config -> Local Station

The dialog window has the following controls:

- Name – This is the name of the test station. This name will be used as the process step name in test reports.
- PN – This is the part number of the test station.
- SN – This is the serial number of the test station.
- Location – This is the factory location identifier for the test station.

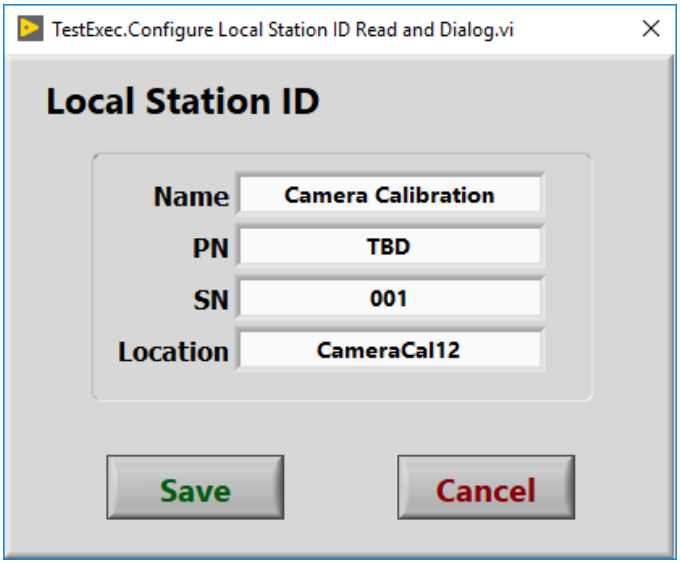


Figure 16 – Local Station Configuration Dialog

5.2.2 Product Section

This section contains information about the product being tested.

```
 [Product]
Name = "HMD"
ProcessStep = "Panel_Cal"
```

The Process Step item is the default process step. See section XX. There is a configuration item to define process steps based on sequence name. If no configuration exists for the current sequence then the default process step will be used. This info will be entered into test reports.

5.2.3 Factory Section

This section contains the location ID of the test station with in the factory:

```
[Factory]
Location = "Station1"
```

5.2.4 Facility Section

This section contains the paths to the Facility Config file and to the Modes Config file:

```
[Facility]
FacilityModes = "\\ServerXXX \Facility\Modes.json"
```

Mode config is the path to a JSON file that identifies the modes and the options for each mode that are used at the factory.

5.2.5 Sequence Section

This section contains the name of the default sequence and it enables/disables auto launch of the sequence.

```
[Sequence]
Default = "Full Test"
```

The default sequence is the sequence that is selected when the application initializes or resets. Certain modes (such as Production) do not allow the operator to select the sequence so this will be the only sequence that can be used.

If no Default sequence is configured, then the first sequence in the sequence folder is selected as the default sequence.

5.2.6 Data Directories

The DataDirs section contains a list of directories that may be used for test result data.

The items in this list may be used by TE and/or by the test station application.

The TE recognizes the following directory named items:

- ResultsLocalDir – by defining this item:
    - If there are no XLINK reports configured (see Test Report section) then a report item will be added with the type XLINK_ResultOnly. At the end of each test run, a XLINK report will be written to this directory.
    - NOTE: This item is obsolete. It is used for backward compatibility to previous test station configurations. Any new test systems should use the Test Report configuration for all test reports.
- ResultsExternalDir – by defining this item:
    - If there are no JSON reports configured (see Test Report section) then a report item will be added with the type JSON_WithFiles. At the end of each test run, a JSON report will be written to this directory.
    - NOTE: This item is obsolete. It is used for backward compatibility to previous test station configurations. Any new test systems should use the Test Report configuration for all test reports.
- ResultsArchiveDir – This defines the folder for saving archive test data.
    - The function 'ArchiveUUTData' will use this folder to archive test result data.
    - If 'ResultsArchive' is not configured, the default folder will be:
        - C:\MTE\[StationName]\Data\Archive

The TE passes the data directory info to the test station. Other data directories may be defined that are used by the test station.

Example Station Config File DataDir section:

```
[DataDirs]
ResultsArchiveDir=    "\\ServerXXX\Utah\StationX\Archive"
```

5.2.7 Route Configuration

Route configuration is used for verifying a product SN is routed to the correct process step.

The station configuration file will have a section named “Routes”. In that section is an item named “RouteList” that has the path to a CSV file as its value.

The CSV file contains a header row (that is ignored) and 2 columns:

Col0 = Route

Col1 = StationName

This file will be read and all routes where the StationName matches the name of this station will be saved to a list. This list is referred to as the Route Destination List for this station.

This section is optional. If there is no Route Configuration entry then the Route Destination List for this station will consist of the station name only.

When a SN is entered, and if the route check feature is enabled (See Mode configuration), a route check will be performed:

1. The Shop Floor Control system will be queried for the inbound process step that is associated with this SN. This is referred to the Route of this UUT.
2. The Route of this UUT will be looked up in the Route Destination List for this station.
    a. If the Route of this UUT is in the Route Destination List then the SN will be accepted.
    b. If the Route of this UUT is in the Route Destination List then the SN will be rejected.

EXAMPLE Configuration entry:

```
[Routes]
RouteList = "C:\MTE\Facility\Circle Test\Circle Test Routes.csv"
```

EXAMPLE CSV File contents:

```
Route,StationName
FW Load, Panel Calibration
Panel Calibration, Panel Calibration
Eyetube Calibration, Eyetube Calibration
Camera Calibration, Camera Calibration
Function Test, Function Test
OQC Function Test, Function Test
```

5.2.8 Sequence Result Directories

The SeqResultDirs section will contain sequence name and folder path pairs.

If a currently running sequence name is in this list, and a report of the type XLINK or XLINK_ResultOnly is configured, that report will be saved to the specified path for the test sequence name rather than the configured report folder.

This provides the ability for test stations to serve as multiple process steps and the output report folder is determined by the sequence name and the Sequence Result Directories configuration.

This section is optional. If there is no Configuration entry then the Route Destination List for this station will consist of the station name only.

Example SeqResultDirs section:

```
[SeqResultDirs]
;SeqName = Folder
TestX = "\\ServerXXX\FactoryLogix\XML\TestX"
TestX_OQC = "\\ServerXXX\FactoryLogix\XML\TestX_OQC"
```

5.2.9 Released SW Directories

The **Released_SW** section will contain information to check and synchronize with released software files. If **ReleasedSWCheckEnable** Is set to TRUE, the Test Executive will perform checks at the interval specified by **ReleasedSWCheckInterval_s** seconds.

The checks will consist of:

Item **ConfigSyncList**

- This method is preferred since the **ConfigFolder** and **SequenceFolder** items are legacy and may not be supported in future releases.
- **ConfigSyncList** will point to a CSV file. The CSV file will contain a list of files/folders to sync from a controlled released software location to the local test station.
- The CSV file contains a 2D array of string with the columns described below. The first row is assumed to be a heading and is ignored. If a string items contains a comma, it must be enclosed in double quotes. If the CSV file is saved by using Excel it will be in the proper format as far as the quotes.
    - Name – this is a user friendly name for the item. It is not used by the Test Executive. It ids intended to be useful for documenting this configuration sync list.
    - Source Path – this is the released file/folder path
    - Destination Path – this is the local file/folder path
    - Sync Option – This indicates how the synced folder will be synced:
        - Sync – this will add the Source contents to the Destination folder. If the items already exist they will be over written. Any Destination items that do not exist in the Source folder will be deleted.
        - Add – this will add the Source contents to the Destination folder. If the items already exist they will be over written. If other items exist in the Destination

folder, they will be ignored. This is the default behavior if the option is other than 'Sync".

Item **ConfigFolder**

- This is the OLD method and although it is currently supported, the **ConfigSyncList** should be used for new test system deployment.
- Folders and files from the specified network folder will be copy them to the station folder: "C:\MTE\[StationName]"
- The Source folder should be named 'Config' so that it will replace config items.

Item **SequenceFolder**

- This is the OLD method and although it is currently supported, the **ConfigSyncList** should be used for new test system deployment.
- Folders and files from the specified network folder will be copy them to the station folder: "C:\MTE\[StationName]"
- The Source folder should be named 'Sequences' so that it will replace config items.

Item **ReleasedSWFile**

- This item specified the path to a CSV file that contains a list of released SW and the released versions.
    - Pull a list of released software items from the CSV file specified by the ReleasedSWFile item.
    - This file has the SW name in the first column and the version I the second column. It has a header row that is ignored.
    - If a SW item detected by TE has a name that matches a name in this list, the TE will indicate a warning if it is not the released version.

Example Released_SW section info:

```
[Released_SW]
ReleasedSWCheckEnable = TRUE
ReleasedSWCheckInterval_s = 3600
ConfigFolder = "\\ServerName\...\Config\"
SequenceFolder = "\\ServerName\...\Sequences\"
ReleasedSWFile = "\\ServerName...\Releases SW Config.csv"
ConfigSyncList = "\\ServerName...\TesterName_ConfigSync.csv"
```

5.2.9.1 Test Limits

The Test Limits directory will contain the limit and result code files for the test station.

This directory will be copied to:

C:\MTE\[StationName]\Config\Test Limits

Any test limit files in the local PC folder will be over written by the released SW files.

5.2.9.2 Sequences

The Sequences directory will sub folders with sequence files.

This directory will be copied to:

C:\MTE\[StationName]\Sequences

Any test sequence files in the local PC folder will be over written by the released SW files.

5.2.10 AdditionalConfig

This section is optional. It contains the path to additional INI files with station configuration information. Any configuration information included in these files will overwrite configuration read from the local configuration file.

```
[AdditionalConfig]
Name1 = "\\ServerXXX\Test Stations\Station Config, StationX.ini"
```

NOTE: For backward compatibility, any section named “Station” is treated as an AdditionalConfig section.

5.3 Facility Modes Config

The facility modes config file is a JSON file and it contains the default mode and a list of all modes and test system behavior options for each mode.

See section 3 for details about mode options.

Modes can be configured from Test Executive when it is in ‘Engineering’ mode.

To edit the mode names and options, select:

Menu -> Config -> Mode Manager

The Mode Configuration Utility will open.

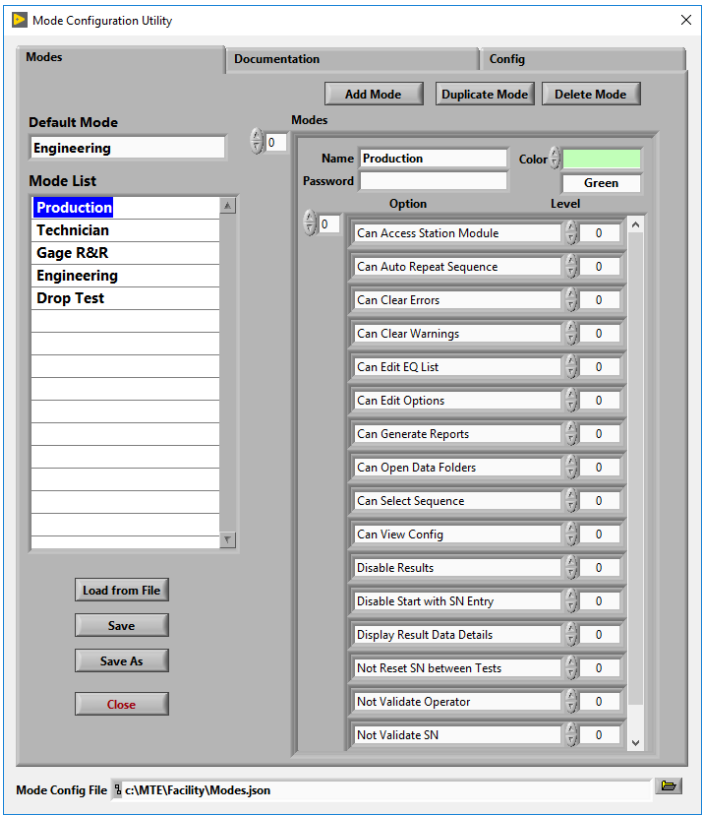


Figure 17 - Mode Configuration Utility

The mode info displayed when this dialog is opened is the mode configuration that Test Executive is configured to use. The name of this path is listed in the Mode Config File field at the bottom of the dialog window.

The Documentation tab has a list of mode options and a description of their function.

Modes can be added using the Add Mode or the Duplicate Mode buttons.

The selected mode's name, password, and options can be edited in the Modes section.

If a mode password is empty, the Test Executive will not prompt for a password before entering this mode.

If a mode password is populates, the Test Executive will prompt for a password before entering this mode.

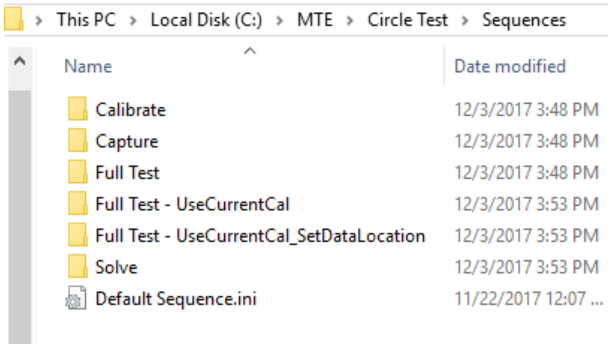
The dialog can be closed by either clicking the Close button or the 'X' in the top right corner.

5.4 Test Sequences

Test Sequences are stored in the folder:

C:\MTE\[StationName]\Sequences

Each sequence is in a separate directory. This architecture allows the ability for a sequence to consist of several files.



In each sequence folder is a file named with the same name as the folder and has a file extension of ".seqx". This is the main sequence file and is run when the sequence is selected.

When TE launches (or resets) it reads all the sub folders in the Sequences folder. It will place this list in the Sequence selector combo box control.

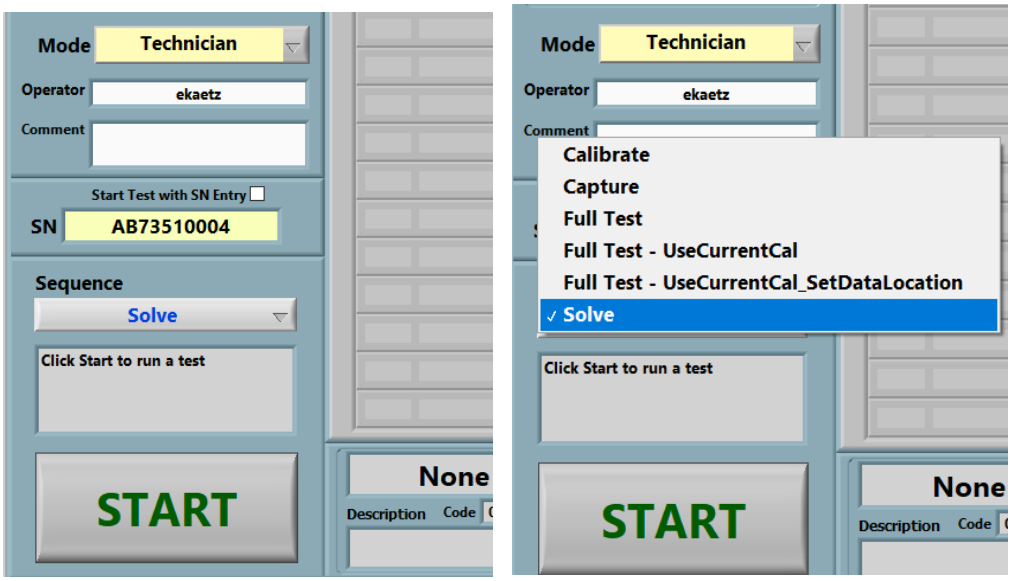


Figure 18 - Sequence Selector

When a sequence is select and the START button is pressed, TE will open the file:

C:\MTE\[App Name]\Sequences\SequenceName\SequenceNameXXXX.seqx

Where XXXX is any text

The sequence file name must begin with the sequence folder name and it must have the file name extension '.seqx'.

There should only be one file matching this description, however, if more than one sequence file matching this description is found, the first file is selected.

NOTE: For backward compatibility, if no files with the file name extension '.seqx' are found, the Test Executive will look for files with a '.seq' or a '.txt' file name extension.

If the sequence file is not found, an error will be reported.

5.5 Test Sequence Repeat

For modes where the option "Can Auto Repeat Sequence" is set to 1 or higher, the auto repeat controls are visible in the header of the Test Executive user interface.

To have a sequence repeat running:

- Turn ON Repeat Seq and set the interval between repeat runs.
- Start the sequence.

Repeating a sequence is useful to validate the stability of a test system and also the stability of the product being tested.

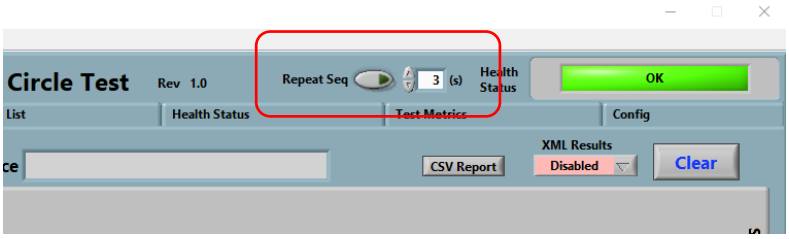


Figure 19 - Auto Repeat Sequence Controls

5.6 Test Limits

The Test executive will look for the test limits file in two locations:

- In the currently running sequence folder: (where SequenceName is the current sequence)

  C:\MTE\[App Name]\Sequences\SequenceName\

- In the default Test limits config folder:

  C:\MTE\[App Name]\Config\Test Limits\

If Test Executive does not find a limit file in the sequence folder, it will look in the default Test Limits folder.

This schema provides the ability to release separate Test Limits for specific sequences.

The Test limits file is named:

Test LimitsXXXX.xml

Where XXXX is any text

The first file found will be selected as the limit file

- The Name of this file less the file extension (.xml) will be reported as the SW item name
- The version inside the file (in tag <FileVersion>) will be listed as the SW version

Test limit files can be generated using the embedded Excel workbook below which has a macro to output to the test limit XML format:


Test Limit Generator

5.7 UUT Barcode Config

When the TE launches it reads the file Barcode Config.ini which is in:

C:\MTE\[App Name]\Config

In this file there is a section named BarcodeFormat:

```
[BarcodeFormat]
Type = "SN_PN_LenDelim"
Length = 12
Delimiter = "TAB"
```

The item Type indicates the type:

- SN_Only
  - The barcode only contains the SN followed by a tab, line feed, or end of line character.
- SN_PN_LenDelim
  - The barcode contains the SN followed by the PN.
  - The SN length is specified in the config file.
- PN_SN_LenDelim
  - The barcode contains the PN followed by the SN.
  - The PN length is specified in the config file.
- SN_PN_StrDelim
  - The barcode contains the SN followed by the PN.
  - The delimiter is specified in the config file.
  - The delimiter is included in the PN. Leading or following white space is trimmed.
- PN_SN_StrDelim
  - The barcode contains the PN followed by the SN. The delimiter is specified in the config file.

- The delimiter is specified in the config file.
- The delimiter is included in the PN. Leading or following white space is trimmed.

The item Length is an integer specifying the length of the first item in the barcode. This item only used for types SN_PN_LenDelim and PN_SN_LenDelim.

The item Delimiter is a string specifying the delimiter between the items in the barcode. This item only used for types SN_PN_StrDelim and PN_SN_StrDelim. Since certain special characters cannot be represented in INI files the following strings represent the specified special characters:

String	Special Character
TAB	Tab (\t)
CR	Carriage Return (\r)
LF	Line Feed (\n)
EOL or CRLF	End of Line (\r\n)

5.8 File Sync Configuration

The TE has the ability to periodically copy the contents of a source folder to a destination folder. This is useful to make sure that certain configuration and other released files are current on the target test station.

NOTE: The sync feature is intended to be used to sync common and released components form a controlled folder to several test stations. The Local Station.INI and any other files that contain information that is specific to the current test station normally would have local copies on the test station and would not be over written with information that is being synchronized (copied) from another location.

This configuration can be setup using a dialog that opens from the TE Menu:

Config -> File Sync

The dialog window has a list with 0, 1, or more File Sync tasks defined. To add an item to this table, start to enter data to an empty record (greyed out). To remove a File Sync task, right click on a task and select “Delete Element” from the shortcut menu.

Each File Sync Task has the following items:

- Name – This is a name for the task. It is not used by the TE. It is included for the user to document the purpose of the task.
- Interval – This is the interval in seconds that the file sync operation will be performed. File sync will only be performed when the test station is in between tests.
- Source Path – This is the source folder path.
- Destination Path – This is the destination folder. Items form the source fielder will be copied to this folder. Any items with the same name will be over written.

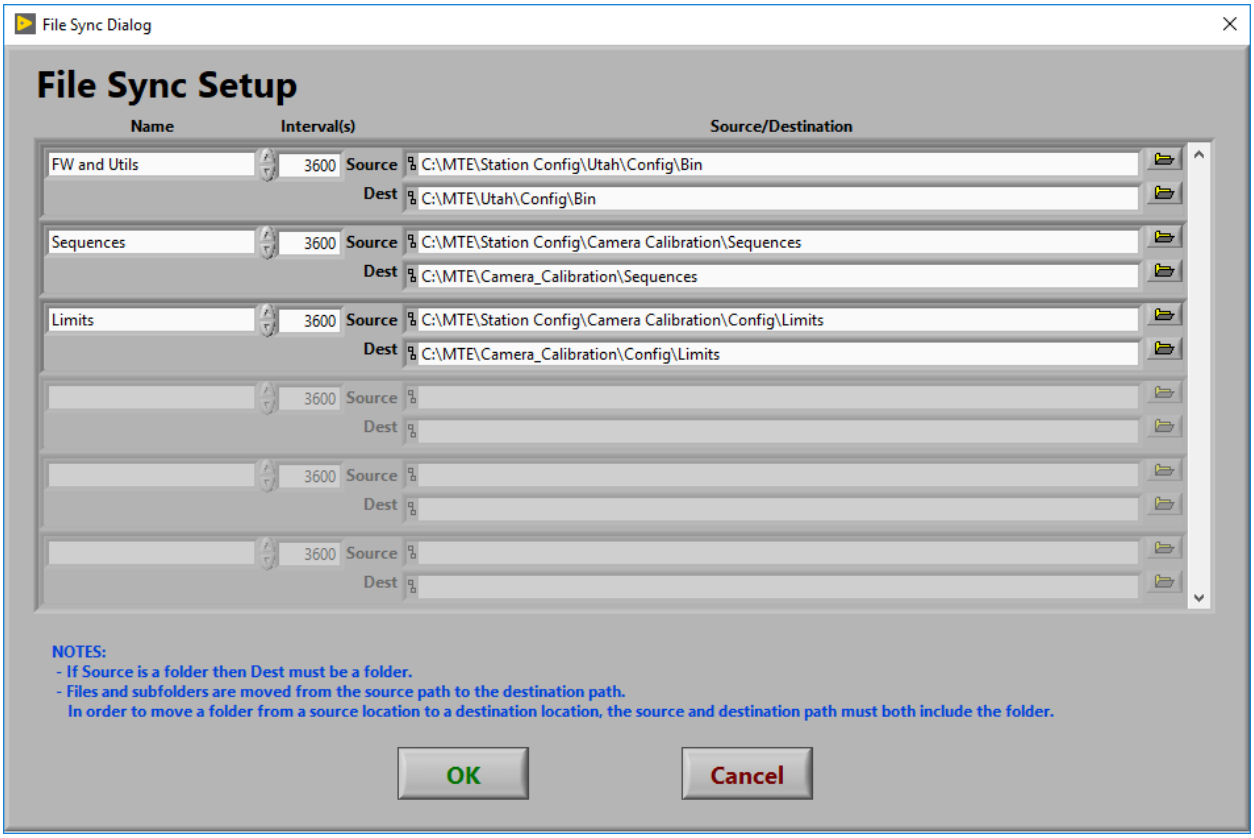


Figure 20 – File Sync Configuration Dialog

File Sync configuration is stored in the file:

C:\MTE\[Station Name]\Config\File Sync Config.csv

This file is in CSV format. The first row is a header. The columns are:

- Name
- Interval(s)
- Source
- Destination

5.9 Test Report Configuration

The TE can be configured to produce 1 or more test reports. These reports will be produced when a test sequence has completed.

The report configuration can be setup using a dialog that opens from the TE Menu:

Config -> Test reports

The dialog window has a list with 0, 1, or more Test Reports defined. To add an item to this table, start to enter data to an empty record (greyed out). To remove a File Sync task, right click on a task and select “Delete Element” from the shortcut menu.

Each Test Report has the following items:

- Name – This is a name for the report. It is not used by the TE. It is included for the user to document the purpose of the task.
- Type – This is the format of the report. The available types are:
  - CSV – this is a table in a CSV file format. The top several rows contain test setup information and the following rows contain test result data.
  - CSV_WithConfigAndData – this report consists of 2 CSV files. One file contains test setup information (Header) and the other file contains test data.
  - XLINK – This is an XML file following the XLINK schema. This report contains test setup and all measured data items.
  - XLINK_ResultOnly – This is an XML file following the XLINK schema. This report contains test setup and only the overall test result. It does not include all the test data items.
  - JSON – This report is JSON file format described by TBD.
  - JSON_WithFiles - This report is a ZIP file that includes a JSON file format described by TBD and a subfolder named “files” with one or more test data files.
- Output Folder – This folder is where the test reports will be written to.

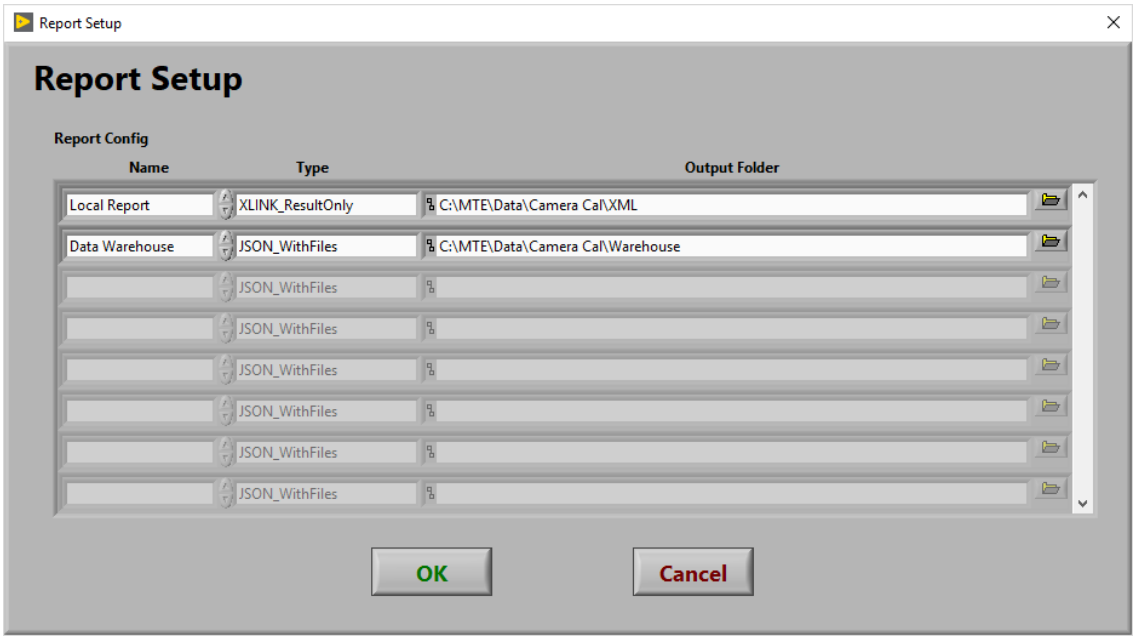


Figure 21 – Test Report Configuration Dialog

Test Reports configuration is stored in the file:

C:\MTE\[Station Name]\Config\Reports Config.csv

This file is in CSV format. The first row is a header. The columns are:

- Name
- Type
- Output Folder

6 Working folders

The TE uses a standard location on the PC for its working folders. The TE will pass the folder locations to the Test Station Module so that it uses those locations also. This makes it easy to locate files for test system maintenance purposes.

6.1 Root Working Folder

The following location is the root working folder:

C:\MTE\[StationName]\

EXAMPLE:

For the test station named “Circle Test” the working folder will be:

C:\MTE\Circle Test\

6.2 Configuration Folder

Test station configuration items are stored in:

C:\MTE\[StationName]\Config\

6.3 Test Sequences Folder

Test station sequences are stored in:

C:\MTE\[StationName]\Sequences\

6.4 Test Report Folder

Test station local reports are stored in:

C:\MTE\[StationName]\Reports\

Only reports generated from the user interface are stored here.

Regular reports results are sent to the configured folder for being pulled into a database.

6.5 Test Log Folder

Test station local reports are stored in:

C:\MTE\[StationName]\Logs\

This folder contains logs generated by test executive to provide maintenance and debugging information.

Log files are named by the day they were generated. Each day new sets of log files are created. Test Executive will delete files older than a specific length of time. The log file retention time is configured.

7 Logging

The TE logs major events. In menu there is a section named 'Log'. When Log-> Event Log is selected, the event log will open.

The event log has a text box that displays the logged events. This window can be left open as TE is operating. The log window will constantly update as new events are logged.

All events are also being logged to a file. The event log file as the current date in its file name. At midnight a new log file is started with the new date in its file name. The files are rotated each day to prevent any file from becoming very large. The log file is stored in the folder:

C:\MTE\[Application Name]\Logs\

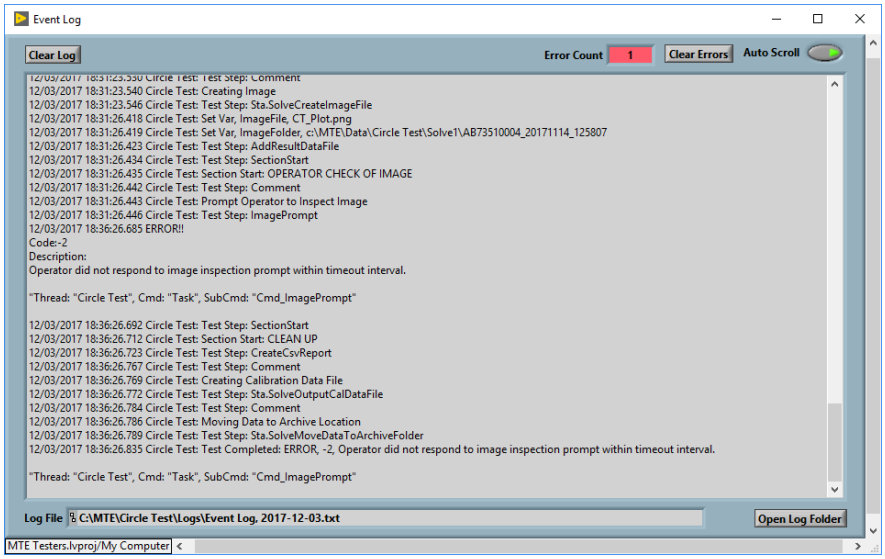


Figure 22 – Event Log

To conserve file space, the TE has a logging level control to set the types of events to be logged. This Logging Level control is on the Config tab -> Options Tab (see Figure 23). The level can be set to:

- Normal – only major events, warnings and errors are logged.
- Verbose – More detailed items are logged
- Debug0 – Reserved for software debugging
- Debug1 – Reserved for software debugging
- Debug2 – Reserved for software debugging

The logging level setting is persistent. It is restored to the last setting when the application is closed and reopened.

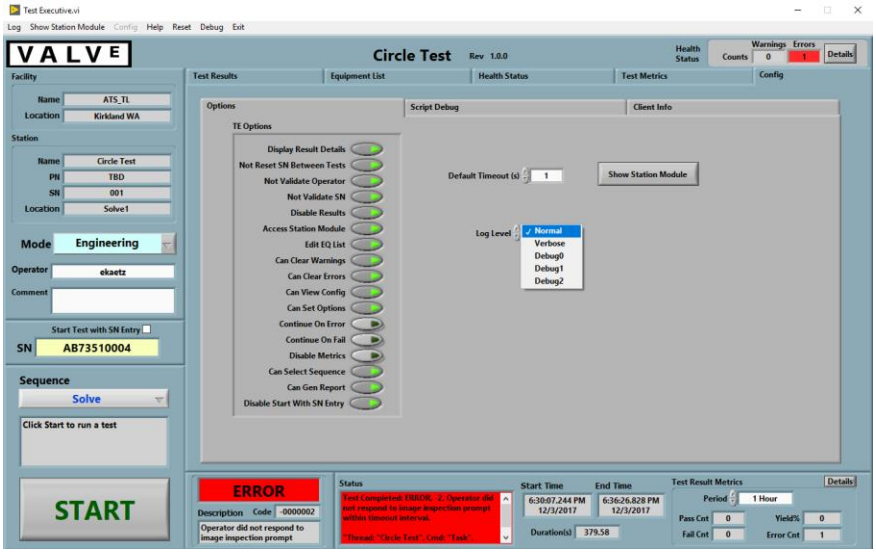


Figure 23 – Logging Level Control

8 Scripting Language

8.1 General

Test executive reads script files and interprets them line by line. The language syntax is described in this document.

Each line may contain a comment, white space, or a single statement. Lines are delimited by the Line Feed character (ASCII code 10).

All arguments in the script language are stored as strings. When they are used they are converted to the data type required by the function or expression such as a number, path, etc..

Statements are processed either by the Test Executive or are forwarded to the Station Control Module to be processed. Any statement that is directed at the station module must begin with the prefix ‘Sta.’. If they do not begin with the ‘Sta. prefix, the Test Executive will attempt to process it and an error will be reported if it is not a recognized function.

This document describes the statements that are supported by the Test Executive. See the specific user manual for the test station for details about script statements that are supported by the station.

Test Executive performs syntax checks to verify all statements in the script are formed correctly.

8.2 White Space

The following characters are considered white space:

- Line Feed also referred to as New Line or Line Break, End of Line (ASCII code 10)
- Carriage Return (ASCII code 13)
- Space (ASCII code 32)
- Tab (ASCII code 9)

Any white space before and after a statement is ignored.

Any lines of the script that contain only white space is ignored.

8.3 Case Insensitivity

All variable names and function names are case insensitive.

8.4 Comments

Lines that begin with a single apostrophe (“’”) or a semicolon (“;”) are treated as comments and are ignored. Text on a line after a semicolon “;” is treated as a comment and is ignored.

Comments can occupy a whole line or they can occupy the end of a line past a statement.

EXAMPLES:

```
'This whole line is a comment
;This whole line is also a comment
FunctionX(InputA) ; This part is a comment
A = 100 ;This part is a comment
```

8.5 Script Version Information

When a test script is run, the Test Executive reads the version information form the top 3 commented lines of the script file. The lines must contain:

; Name = SEQUENCE_NAME

; PN = PART_NUMBER

; Version = VERSION

Where:

- SEQUENCE_NAME is a string representing the name of the sequence.

If this is omitted, then Test Executive will use the name of the sequence folder.

- PART_NUMBER is a string representing the part number of the sequence.
- VERSION is a string representing the version of the sequence. The format of the version of the sequence should be consistent with versions used for other software used on the test systems.

EXAMPLE:

```
; Name = Camera Calibrate
; PN = M000084-01
; Version = 1.0.1
```

Test Executive collects the version information of sequences and other software components and includes this information in test reports.

8.6 Statements and Recursion

A statement can only contain a single operation. Recursive statements are not supported. The arguments of a statement function must be constants or variables. They cannot contain other statements such as another function or math expression. An example of a recursive expression that is **NOT** supported is:

Function1(Expression1(input1, Expression2(A, B))

In the above example, the interpreter would need to solve Expression2 before Expression1 then Expression1 before Function1. This recursive behavior is not supported in this version of Test Executive but may be supported in future releases.

8.7 Constants

All constants are strings. Arguments to a function or expression may be variables or constants.

A constant is converted to the necessary data type based on the way it is being used. If it cannot be converted to the correct type an error will be indicated. An example is when a string that is not a number is placed as an argument to a function that requires a number.

8.8 Variables

Variables are assigned using assignment statement:

VariableName = Value

The variable VariableName will be assign Value.

Variable naming conventions:

- Must start with a letter
- May contains alpha numeric characters (i.e. letters and numbers)
- May contain the non-alpha numeric characters:
  - "_" Underscore
  - "." Period

All variable values are stored as strings data types. They are converted to other data types when they are used in expressions or functions that require other data types.

Strings may be enclosed in double quotes, but the quotes are not required.

Commas are used as delimiters for functions. An input to a function cannot be a string that contains a comma. Variables can be assigned to strings that contain commas. A variable that is assigned a string containing a comma may be used as an argument to a function.

EXAMPLE:

```
X = "The Sky is Blue"
     ;X is assigned to the string "The Sky is Blue"
X = The Sky is Blue
     ;X is assigned to the string "The Sky is Blue"
X = 1034.5
     ;X is assigned to the string "1034.5"
```

It is acceptable (but not recommended) to use a variable before it has been assigned a value. It will have the value of an empty string.

To use a variable, it must be entered in the script with a prefix character of "$".

EXAMPLE:

```
X = 5           ;X is assigned "5"
Y = 7           ;Y is assigned "7"
Z = $X + $Y     ; Z = 7 + 5 = "12"

EnableFunctionX = 0
If($EnableFunctionX = 1)
  FunctionX()
End If
```

Variables may be placed inside the string of a constant. When the statement is run the variable will be replaced with its string value.

EXAMPLE:

```
X = "Today is Friday"
Comment("Variable X = $X")
;The comment that is displayed is: Variable X = Today is Friday
Y = 10
```

```
Comment("Test $Y is being performed")
;The comment that is displayed is: Test 10 is being performed
```

8.9 Arrays

Arrays are variables with an index value in brackets at the end of the variable name. The index value must be an integer. It can be hard coded to an integer or a variable can be used to provide the array index. When hardcoding an index, do not include leading zeros.

Arrays are stored the same way other variables are stored – as strings. The data value in the array element will be converted to the appropriate type when it is used.

EXAMPLE Assigning Array Elements:

```
MyArray[0] = 5
MyArray[1] = "ProductX"
MyArray[2] = "c:\MTE\Data\ProductX"
```

EXAMPLE Assign an Array Using a For Loop:

```
For i = 1 to 4
     X[$i] = $i * 2
Next
```

The above example will assign the variables:

```
X[1] = 2
X[2] = 4
X[3] = 6
X[4] = 8
```

8.10 Numbers

When a constant or a variable is used as a number, it is converted to a number it is a 64-bit floating point type. When the result of a math expression is stored to a variable, it is converted to a string representing a floating-point number with 6 digits of precision.

8.11 Arithmetic Expressions

Only expressions with 2 terms and the following operations are supported:

Addition:	VariableX = Value1 + Value2
Subtraction:	VariableX = Value1 - Value2
Multiplication:	VariableX = Value1 * Value2
Division:	VariableX = Value1 / Value2

In each of the above expressions, the first term must be a variable. The other 2 terms can be either variables or constants.

There must be at least one space delimiter before and after the operators: =, +, -, *, /

8.12 Flow Control

8.12.1 Comparison Operators

The following is a list of valid logic operators:

Operator	Description
=	Equal
!= or <>	Not Equal
<	Less Than
<=	Less Than or Equal To
>	Greater Than
>=	Greater Than or Equal To

Only 2 items (variables or constants) can be compared at a time. Compound logic expressions are not supported.

8.12.2 If Statement

The syntax for 'If' statements are:

```
If(Value1 ComparisonOperator Value2)
      Statement
      Statement
Else
      Statement
      Statement
End If
```

The Else statement is optional.

If the expression in the IF statement brackets is True the statements between the IF and the Else will be executed. If there is no Else statement then the statements between the IF and the End If will be executed.

If the expression in the IF statement brackets is False the statements between the Else and the End If will be executed. If there is no Else statement then the execution will continue with statements that are after the End If.

EXAMPLE:

```
;** Skip analyze if passing
If($Failing = 1)
  Comment("Analyzing Data")
  SendDataToSta()
  Sta.AnalyzeData()
End If
```

If statements can be nested in other flow control statements. Every If statement must have an End If statement.

8.12.3 While Statement

The syntax for 'While' statements are:

```
While(Value1 ComparisonOperator Value2)
      Statement
      Statement
End While
```

If the expression in the While statement brackets is True the statements between the While and the End While will be executed.

When End While is reached, the execution will jump back to the While statement.

If the expression in the While statement brackets is False the execution will jump to the statements after the End While statement.

EXAMPLE:

```
X = 0
While($X < 100)
      Comment("X = $X")
      X = $X +10
End While
```

While statements can be nested in other flow control statements. Every While statement must have an End While statement.

8.12.4 For Statement

The syntax for 'For' statements are:

```
For CounterVariable = StartNum to EndNum, Step
      Statement
      Statement
Next
```

The Step parameter is optional. If it is omitted, the Step value of 1 will be used.

The Step argument may be positive or negative.

If it is Positive, the EndNum must be greater than the StartNum argument.

If it is negative, the EndNum must be less than the StartNum argument.

Execution Description:

- The first iteration of the For statement will assign the value of the CounterVariable to StartNum
- Statements between the For and the Next statement will be executed.
- When the Next statement is reached, execution will jump back to the For statement.
- The CounterVariable will be incremented or decremented by the Step value.
    - If the Step is positive and the CounterVariable <= EndNum
        - Statements between the For and the Next statement will be executed
        - Otherwise the statement past Next will be executed
    - If the Step is negative and the CounterVariable >= EndNum
        - Statements between the For and the Next statement will be executed
        - Otherwise the statement past Next will be executed

EXAMPLE:

```
;X will be incremented from 0 to 100 with steps of 10
For X = 0 to 100, 10
      Comment("X = $X”)
Next


;Y will be incremented from 0 to 10 with steps of 1
For Y = 0 to 10
      Comment("Y = $Y”)
Next
```

For statements can be nested in other flow control statements. Every For statement must have a Next statement.

8.12.5 GoTo Statement

The syntax for ‘GoTo’ statements are:

```
GoTo(LabelName)
. . .
. . .
. . .
LabelName:
```

The GoTo statement redirects the flow to a label.

A label is a name that ends with a colon.

8.12.6 Delay(Seconds), Wait(Seconds)

This function will suspend execution for the number of seconds specified in the argument.

The function Delay or Wait perform the same actions (they are interchangeable).

Argument	Data Type	Required	Description
Seconds	Float	Yes	Seconds to delay

8.12.7 End(), Exit()

This function will cause the sequence to terminate.

The sequence behaves as if it completed executing its last line. The test will PASS if no failure or error has occurred.

8.12.8 CancelTest()

This function will cause the sequence to terminate.

No report will be generated.

8.12.9 RepeatMe(Seconds)

If seconds is greater than or equal to 0, this function will cause this sequence to be repeated the specified seconds after it has completed.

If seconds is less than 0, this function will cause this sequence to not be repeated.

Argument	Data Type	Required	Description
Seconds	Float	Yes	Seconds to wait before repeating

8.12.10 SetAbort(Description)

This function will cause the test to abort.

Argument	Data Type	Required	Description
Description	String	Yes	Explanation for the reason the test abort action was invoked

8.12.11 SetError(Code, Description)

This function will cause an error to be reported. This will cause the overall result to be ERROR.

Typically, a test is set to stop on an error.

Argument	Data Type	Required	Description
Code	Integer	Yes	Negative value. This value should be based on a protocol for ERROR codes defined by the test engineering department.
Description	String	Yes	Description of error.

8.12.12 SetFail(Code, Description)

This function will set the sequence to fail.

Argument	Data Type	Required	Description
Code	Int	Yes	A failure code
Description	String	Yes	Description of the failure

8.12.13 SetStopOnFail([Enable])

This function will turn Stop on failure ON or OFF.

If no argument is provided then stop on failure will be set to Enabled.

Argument	Data Type	Required	Description
Enable	Number	Yes	1 = Stop on Failure 0 = Continue on failure Empty = Stop on Failure

8.13 Functions

Functions are formed by a name followed by comma delimited arguments enclosed in brackets:

FunctionName(Arg1, Arg2, Arg3)

Prefixes are used to route functions to the module that will process the function. A prefix is a name followed by a period. Functions that do not have a prefix are processed by the Test Executive application. The prefix "Sta." will route the function to the Station Control Module. Other prefixes may be supported to route the function to other modules in the test system. See the user manual for the target test system to learn of other supported functions and the routing prefix.

The functions described in this section are processed by the Test Executive application.

8.13.1 Test Results and Status

8.13.1.1 Measurement(Category, Name, Value,[Units, LimType, LL, UL, Status, Code, Description])

This function will record test result data.

If the name is in the list of limits that are configured for this test station, the following arguments will be replaced with those in the limits table:

LimType, LL, UL, Status, Code

The arguments are:

Argument	Data Type	Required	Description
Category	String	Yes	A name that is used to sort and filter the results
Name	String	Yes	The name of the result
Value	String/Number	Yes	The value to record.
Units	String	Optional	The units of the value. This is optional and may be left empty.
LimType	String	Optional	The comparison type for using the limits if applicable. See Table 2 for a list of LimTypes. If this result name is in the Limit file, this will be overwritten by the status calculated by the limit file comparison.
LL	String/Number	Optional	Lower Limit. If this result name is in the Limit file, this will be overwritten by the value in the limit file.
UL	String/Number	Optional	Upper Limit. If this result name is in the Limit file, this will be overwritten by the value in the limit file.
Status	String	Optional	PASS, FAIL, ERROR, or LOG. It will be set to LOG if it is omitted. If this result name is in the Limit file, this will be overwritten by the status calculated by the limit file comparison.
Code	Integer	Optional	0 for PASS and LOG. Positive for VALUE. Negative for ERROR. This value should be based on a protocol for FAIL and ERROR codes defined by the test engineering department.

			If this result name is in the Limit file, this will be overwritten by the status calculated by the limit file comparison.
Description	String	Optional	Description of fail or error. This parameter is optional. It will be set to "" if it is omitted. If this result name is in the Limit file, this will be overwritten by the status calculated by the limit file comparison.

Table 2 - Limit Type

Limit Type	Description	Notes
None	No limits are applicable	Leaving the field empty will also indicate that no limits are applicable.
EQ	Equal To	use LL
NE	Not Equal To	use LL
GT	Greater Than	use LL
GE	Greater Than or Equal To	use LL
LT	Less Than	use UL
LE	Less Than or Equal To	use UL
GTLT	Greater Than LL and Less Than UL	use LL and UL
GELE	Greater Than or Equal To LL and Less Than or Equal To UL	use LL and UL
GELT	Greater Than or Equal To LL and Less Than UL	use LL and UL
GTLE	Greater Than LL and Less Than or Equal To UL	use LL and UL
LTGT	Less Than LL and Greater Than UL	use LL and UL
LEGE	Less Than or Equal To LL and Greater Than or Equal To UL	use LL and UL
LEGT	Less Than or Equal To LL and Greater Than UL	use LL and UL
LTGE	Less Than LL and Greater Than or Equal To UL	use LL and UL
Contains	Contains the string listed in the LL	use LL
NotContain	Does Not Contain the string listed in the LL	use LL

8.13.1.2 AddEquipmentItem(Name,Mfg,PN,SN,SW_FW,Note)

This function will append an equipment item to the equipment list. This list is added to the test report.

Argument	Data Type	Required	Description
Name	String	Yes	Name of the equipment item
Mfg	String	Yes	Manufacturer

PN	String	Yes	Part Number
SN	String	Yes	Serial Number
SW_FW	String	Yes	SW and/or FW version
Note	String	Yes	Any notes regarding the equipment.

8.13.1.3 AddNumericResult(Category, Name, Value,[Status, Code, Description])

This function will append a numeric result to the test report. The arguments are:

Argument	Data Type	Required	Description
Category	String	Yes	A name that is used to sort and filter the results
Name	String	Yes	The name of the result
Value	String	Yes	A numeric value
Units	String	Yes	The units of the value.
Status	String	Optional	PASS, FAIL, ERROR, or LOG. It will be set to LOG if it is omitted.
Code	Integer	Optional	0 for PASS and LOG. Positive for VALUE. Negative for ERROR. This value should be based on a protocol for FAIL and ERROR codes defined by the test engineering department. If this result name is in the Limit file, the value will be compared against the limits and if it fails the code and will be populated based on the limit file fault code.
Description	String	Optional	Description of fail or error. This parameter is optional. It will be set to “” if it is omitted. If this result name is in the Limit file, and the value fails the limit test, this description will be populated based on the limit file description.

8.13.1.4 AddResultDataFile(Folder,FileName, [Category,Name])

This function will append a data file to the report that is sent to the Data Warehouse.

Argument	Data Type	Required	Description

Folder	String	Yes	Folder of the data file. Typically this is the UUTDir.
FileName	String	Yes	The name of the data file
Category	String	No	This is the category attribute of this data item. If this argument is not included then the category "File" will be sued by default.
Name	String	No	This is the Name attribute of this data item. If this argument is not included then the name of the file (less extension) will be used by default.

8.13.1.5 ArchiveUUTData(SpecialInstruction, . . SubFolder, . . .)

This function will move the current UUT Folder to the configured archive folder location.

There may be one or more argument. Each argument specifies a subfolder within the Archive folder to write the data to.

The arguments may be special instructions which are key words used as arguments. These key words are:

AppendSeq – this will cause the sequence name to be appended to the name of the UUT folder.

AppendResult – this will cause the PASS or FAIL result name to be appended to the name of the UUT folder.

The arguments may be strings or variables. If the argument is a string, it will be used as a subfolder name. If the argument is a variable, it will be replaced with the value at runtime. A variable may be used to indicate the date or mode.

Example Archive function command:

```
ArchiveUUTData(AppendSeq,AppendResult,$Mode, $Date)
```

8.13.1.6 ClientAttribute(Folder, FileName)

This function will append an attribute (setup item) of the test system that will be recorded to the test report.

Argument	Data Type	Required	Description
Category	String	Yes	A name that is used to sort and filter the results
Name	String	Yes	The name of the attribute
Value	String	Yes	A string value that represents the value of this attribute

8.13.1.7 Comment(String)

This function will display text in the Status text bod on the Test Executive Front Panel.

Argument	Data Type	Required	Description
String	String	Yes	Text to display

If the Test Executive logging level is set to Verbose or higher, this comment will be recorded in the event log.

8.13.1.8 CreateCsvReport(Folder)

This function will generate 2 CSV files and will place them in the folder specified by the argument.

The files are named:

- SerialNum_Date_Time, Sequence Name, Report Header.csv
- SerialNum_Date_Time, Sequence Name, Report Data.csv

Where

- SerialNum is the serial number of the UUT
- Date is ion the format YYYYMMDD
- Time is in the format HHMMSS

The Report Header file contains the overall result and test setup information.

The Report Data file contains numeric and attribute data in a single table.

Argument	Data Type	Required	Description
Folder	String	Yes	Folder to place the CSV report in

8.13.1.9 SectionStart(SectionName), SubSectionStart(SectionName), SubSectionEnd()

The test report has headings with step numbers that are similar to heading numbers of a document. A test section is the highest level.

The SectionStart function will:

- Increment the highest level heading counter
- Append an attribute in the report to represent a new section heading.
    - The heading name will displayed in upper case characters
    - The first section of a report will be step ‘1.0’.
    - The next section of a report will be step ‘2.0’ etc.

Data that is appended to the report will have step numbers that increment the counter of the level below the current section.

- The first data item of section 1.0 will be step 1.1.
- The second data item will be step 1.2 etc.

The SubSectionStart function will:

- Will append an attribute in the report to represent a new sub section heading.
  - The heading name will displayed in upper case characters
  - This heading will be at the level of a data item
- Will set the current level to one level lower.
  - All data added to the report will be at the level below the current section level

The SubSectionEnd function will:

- Will set the current level to one level higher

The SectionStart and SubSectionStart argument is:

Argument	Data Type	Required	Description
SectionName	String	Yes	Name of the section

EXAMPLE:

Script Statement	Report Step	Report Item Name
SectionStart (“Initializing”)	1.0	INITIALIZING
AddAttribute(“XX”,Data1,….)	1.1	Data1
AddAttribute(“XX”,Data2,….)	1.2	Data2
SectionStart (“TestX”)	2.0	TESTX
AddAttribute(“XX”,Data3,….)	2.1	Data3
AddAttribute(“XX”,Data4,….)	2.2	Data4
SubSectionStart(“TestY”)	2.3	TESTY
AddAttribute(“XX”,Data5,….)	2.3.1	Data5
AddAttribute(“XX”,Data6,….)	2.3.2	Data6
SectionName(“Clean Up”)	3.0	CLEAN UP

8.13.1.10 SendDataToSta()

This function will cause the Test Executive to send result information to the Station Module. It has no arguments.

The Test Executive knows the test limits, but the station module does not.

The station module has knowledge about the station tests, but the Test Executive does not.

This function is used together with a station specific function (see specific Station User Manual) that will analyze the data and return a modified overall result to the Test Executive, so the test report will have an overall result that provides better detail of how the test is failing.

8.13.1.11 SequenceConfig(Name, Value)

This function provides a mechanism for the test script to pass sequence setup parameters to the Test Executive. Sequence setup info is included in the header of the test report.

Argument	Data Type	Required	Description
Name	String	Yes	Name of the setup item
Value	String	Yes	Value of the setup item

8.13.1.12 SetDisableResults(OneZero)

This function will enable or disable result recording for the test sequence.

If the argument is 1 the results will be disabled, and no report will be generated for this test run.

If the argument is 0 (or anything but 1) the results will be enabled, and a report will be generated based on the Disable Results option setting.

This function is useful if a test is canceling itself based on a parameter or result and it is not desired to have results reported.

Argument	Data Type	Required	Description
OneZero	String	Yes	1 = Disable Results Other = Enable Results

8.13.2 Operator Prompts

8.13.2.1 DataPrompt(Prompt, [VariableName, Type])

This function will cause the Test Executive to prompt the operator to enter data.

There are 3 types of prompt:

- String Data
  - Visible items are: Instructions, a string control and the OK button.
  - The prompt ends when data is entered and terminated with the Enter key or when the OK button is clicked.
  - The entered string is returned to the script.
- OK
  - Visible items are: Instructions and the OK button.
  - The prompt ends when the OK button is clicked.
- Yes/No
  - Visible items are: Instructions, the YES and NO buttons.
  - The prompt ends when the YES or NO button is clicked.
  - The string “YES” or “NO is returned to the script based on the selected button.

It is up to the test sequence to process the data entered.

Argument	Data Type	Required	Description
Prompt	String	Yes	Message to display
VariableName	String	Optional	Variable name of the data top return If no variable name is provided the text will be written to a variable named ‘'OperatorDataInput'’
Type	Integer	Optional	Type of prompt to show. 0 or empty = Prompt for string data 1 = OK prompt 2 = Prompt for Yes or No

8.13.2.2 ImagePrompt(Prompt, FileOrFolder, [FileName, ResultName)

This function will cause the Test Executive to prompt the operator to approve or disapprove an image.

The operator will be prompted with the text string in the first argument of this function. A YES and a NO button will be available. If the operator clicks YES this test action will PASS. If the operator clicks NO this test action will FAIL.

An attribute result will be recorded for this test action. The attribute will be named by the string contained in the argument ResultName.

Argument	Data Type	Required	Description
Prompt	String	Yes	The prompt for the operator

FileOrFolder	String	Yes	If the File argument is blank or omitted, this will be treated as the full path of the image file. If the File argument is not empty, this will be treated as the folder of the image file.
FileName	String	Yes	The name of the image file
ResultName	String	Yes	The name of the result attribute to record

8.13.3 Math

8.13.3.1 AbsVal(VarName)

This function will take the value of the provided variable and will modify it to make it positive.

EXAMPLE:

For variable X = (-)12.34

AbsVal(X) will modify the value of X to be 12.34.

Argument	Data Type	Required	Description
VarName	String	Yes	The variable to be modified

8.13.3.2 Int(VarName)

This function will take the value of the provided variable and will modify it to make it an integer.

EXAMPLE:

For variable X = 3.14

Int(X) will modify the value of X to be 3.

Argument	Data Type	Required	Description
VarName	String	Yes	The variable to be modified

8.13.3.3 QuotRemainder(X, Y)

This function will calculate the quotient and remainder of X/Y then will produce 2 varibales with these results:

ValRemainder = Remainder of X/Y

ValQuotient = Quotient of X/Y

EXAMPLE:

For input X = 5 and Y = 2

ValQuotient will be 2

ValRemainder will be 1

Argument	Data Type	Required	Description
X	Number	Yes	A number or variable that has a value which is a number.
Y	Number	Yes	A number or variable that has a value which is a number.

8.13.4 Strings and Paths

8.13.4.1 PathBuild(OutputVarPath, BasePath, FolderName1, FolderName2, ... FolderNameN)

This function builds a path.

The first argument is the variable name where the result path will be written to.

The following arguments are subfolders. There is no limit to the number of subfolders provided. The last argument can be a subfolder or a file name.

Argument	Data Type	Required	Description
OutputVarPath	String	Yes	Variable name to write the built path into.
BasePath	String	Yes	This is the starting path.
FolderNameN	String	1 or more	These are added to the base path to build the result path. The last item can be a folder or a file name.

8.13.4.2 PathStrip(OutputVarBasePath,OutputVarName,InputPath)

This function splits a path.

The first argument is the variable name where the base path will be written to.

The second argument is the variable name where the stripped folder or file name will be written to.

The third argument is the path to be stripped.

Argument	Data Type	Required	Description
OutputVarBasePath	String	Yes	Variable name to write the base path into.
OutputVarName	String	Yes	Variable name to write the stripped folder or file name into.
InputPath	String	Yes	This is the path to be stripped.

8.13.4.3 StrConcat(OutputVar,Str1, Str2, ...StrN)

This function concatenates strings.

The first argument is the variable name where the concatenated string will be written to. The following arguments are strings to concatenate. There should be at least two string items to concatenate. There is no limit to the quantity of string arguments to concatenate (other than the PC memory resources).

Argument	Data Type	Required	Description
OutputVar	String	Yes	Variable name to write the concatenated string into.
StrN	String	2 or more	These are the strings to concatenate.

8.13.5 Configuration

8.13.5.1 ReadINIConfig(Folder,FileName,[Section1,ItemName1,Section2, ItemName 2,...])

This function reads data from an INI file.

If the Folder parameter is empty, then the config folder for the application will be used.

The INI file at the specified folder and name will be opened.

If the config file exists then:

- A variable named 'ConfigFileExists' will be created and will be set to 1.
  - This variable can be checked by the script to determine the next action
- A variable will be created for each Section and ItemName pair. The variables will be named:

  Section1.ItemName1

  Section2.ItemName2

  . . . .

- The values at the Section and Item Name pairs will be read and written to the corresponding variable. These variables are available to be processed by the script.

If the config file does not exist then:

- A variable named 'ConfigFileExists' will be created and will be set to 0.
    - This variable can be checked by the script to determine the next action
- All variables will be created but will be returned as empty string.

8.13.6 Timers

8.13.6.1 SetTimeout(Seconds)

The Test Executive has a timeout that it applies to each function call. This is to abort in case a station module would get stuck trying to perform a function.

The SetTimeout function sets the timeout used for function calls. The default time out is 1 second. If a function call is anticipated to take longer, the script should adjust the timeout as necessary.

Argument	Data Type	Required	Description
Seconds	Float	Yes	Seconds to delay

8.13.6.2 TimerGet(VarElapsedTime ,[TimerNum])

The script has a feature to use up to 10 timers.

The timers are initiated to the current time by the TimerInit function.

The TimerGet function will read the elapsed time (in seconds) of the time since it was initiated.

All timers are initiated at the test sequence start. Checking the timer without initiating it will return the time since the test sequence started.

Argument	Data Type	Required	Description
VarElapsedTime	String	Yes	This is the name of the variable that will return the value of the timer.
TimerNum	Int 0 to 9	No	Timer number. If no parameter is provided, then timer 0 will be selected.

8.13.6.3 TimerInit([TimerNum])

The script has a feature to use up to 10 timers.

The timers are initiated to the current time by the TimerInit function.

The TimerGet function will read the elapsed time (in seconds) of the time since it was initiated.

All timers are initiated at the test sequence start. Checking the timer without initiating it will return the time since the test sequence started.

Argument	Data Type	Required	Description
TimerNum	Int 0 to 9	No	Timer number. If no parameter is provided, then timer 0 will be selected.

8.13.6.4 TimeLog(Name,[Category])

This is used for characterizing and debugging test times.

It creates a Numeric Result item with the value = time since the test start.

The result name is the name provide in the first parameter

The result category is the name provided in the second parameter. If this is empty, the category will be set to "TimeLog".

This function also creates a variable names “TimeLog” with the value equal to the time in seconds since the start of the test. This variable is available to the script.

8.13.7 Operating System

8.13.7.1 DeviceEnumCheck(VID,PID,[VariableName])

This command is used to check if a USB device is enumerates. It returns a variable with the status:

1 = enumerated

0 = is not enumerated

Argument	Data Type	Required	Description
VID	String	VID and/or PID must be provided.	The VID of the device to check. If this is empty, then only the PID match will be checked.
PID	String	VID and/or PID must be provided.	The PID of the device to check. If this is empty, then only the VID match will be checked.
VariableName	String	Optional	The name of the variable to use for returning the enumeration status. If this is not provided the status will be returned in the variable named: “DeviceIsEnumerated”

8.13.7.2 ProcessStatus(ProcessName, [VariableName])

This command is used to check if a process is running. It returns a variable with the process status:

1 = running

0 = is not running

Argument	Data Type	Required	Description
ProcessName	String	Yes	The name of the process to check. Must match the name as it appears in the Windows Task Manager.
VariableName	String	Optional	The name of the variable to use for returning the process running status. If this is not provided the process status will be returned in the variable named: “ProcessIsRunning”

8.13.7.3 ProcessKill(ProcessName)

This command is used to kill a process.

Argument	Data Type	Required	Description
ProcessName	String	Yes	The name of the process to kill. Must match the name as it appears in the Windows Task Manager.

8.13.8 Misc

8.13.8.1 Exists(VarName)

This function will check if a Variable has been defined.

The variable ‘VarExists’ will be created with the result.

If the variable ‘VarName’ exists, VarExists’ will equal 1

If the variable ‘VarName’ does not exists, VarExists’ will equal 0

Argument	Data Type	Required	Description
VarName	String	Optional	Variable name to be checked.

8.13.8.2 PLCStart()

This command is used when the test interacts with a program logic controller(PLC). This function sends a start command to the PLC to cause it to perform an action required by the test sequence.

8.13.8.3 Print_Label(Print_Text, URL)

This functions sends a HTTP command to a printer to print text.

Print_Label accepts different arguments. This instance of Print_Label will be used if 2 arguments are present.

Argument	Data Type	Required	Description
Print_Text	String	Yes	The text to be printed.
URL	String	Yes	This is the URL to send to the printer. It contains an address and arguments for the printer to perform the desired function. This URL string must contain a field "%PRINT_TEXT%. This will be replaced with the Print_Text argument to this function.

8.13.8.4 Print_Label(Print_Text, File, CfgSection, CfgItem)

This functions sends a HTTP command to a printer to print text.

Print_Label accepts different arguments. This instance of Print_Label will be used if 4 arguments are present.

Argument	Data Type	Required	Description
Print_Text	String	Yes	The text to be printed.
File	String	Yes	The configuration file name or the full path to the configuration file. The configuration file must be an INI file. If the file argument provided is a file name then folder used is "c:\MTE\[Test Station]\Config\"
CfgSection	String	Yes	This is the INI file section name that contains the information for the printer.

CfgItem	String	Yes	This is the item name (in section CfrSection) that contains the URL string to send to the printer. The URL string must contain a field "%PRINT_TEXT%. This will be replaced with the Print_Text argument to this function.

8.13.8.5 XlinkPartInitReport(DestinationFolder,Batch,Name)

This function builds an XML file with data that will initiate a part in the FactoryLogix database.

Argument	Data Type	Required	Description
DestinationFolder	Path	Yes	Specifies the folder to write the file to.
Batch	String	Yes	The batch ID of the part
Name	String	Yes	The name of the part

The file will be named:

SN_Date_Time.xml

Where:

SN is the serial number of the unit under test

Data_Time is the test start time minus one second

EXAMPLE XML File:

```
<?xml version="1.0" encoding="UTF-8"?>

<BarcodeList operator="souk"> <Job assembly="N/A" assemblyRev="N/A" name="Motor Jitter Test" batch="Motor-Jitter-Test" lot="N/A"> <Barcode id="RB7500153BV001889-03.01" scanTime="2018-01-11T11:35:30"/> </Job> </BarcodeList>
```

In the above example file the following property fields (in Yellow) will be replaced based on the test being run and the parameters of the function:

operator:	The test operator
name:	Name (the 3rd function parameter)
batch:	Batch (the 2nd function parameter)
id:	Serial Number of the UUT
scanTime:	Test Start Time minus one second