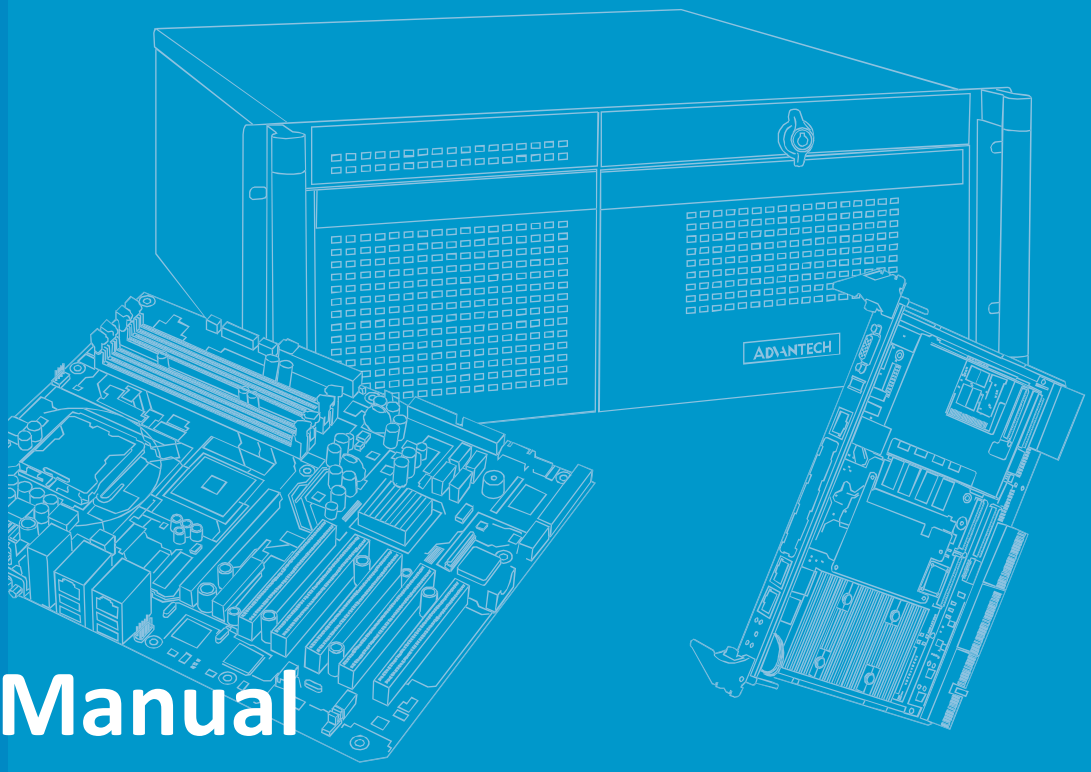



# User Manual

## ADVANTECH

## DIAGNOSTIC FRAMEWORK

Revision 6.0

ADVANTECH

*Enabling an Intelligent Planet*

# Revision History

| Date [mm/dd/yyyy] | Revision | Modifications |
|---|---|---|
| 10/29/2020 | 6.0 | Official release Edition 6.0<br>2.1 step1 to install from binary package<br>4.2.9.2 Format of scripts (valid function id ) |
| 06/05/2020 | 5.0 | Official release Edition 5.0 |
| 05/19/2020 | 4.1 | Added chapter for dui_cfg<br>Updated description of libraries and 3rd-party utilities<br>Updated the table of service list |
| 3/30/2020 | 4.0 | Official release |
| 3/2/2020 | 3.1 | Added Chapter for new diagnostic service: PCI<br>Added JSON output of HTTP interface<br>Updated image and chapter of TUID |
| 1/14/2020 | 3.0 | Official release Ed3.0 |
| 12/18/2019 | 2.2 | Added chapter of troubleshooting Added footnote to pictures/images in TUID chapter |
| 11/28/2019 | 2.1 | Added chapter for TUID<br>Added chapter for troubleshooting<br>Changed the parameter description of lan_func |
| 08/23/2019 | 2.0 | Official release |
| 07/16/2019 | 1.3 | Updated unit of memory size parameter for mem_dmi and mem_test<br>Removed dmidecode<br>Removed RPC<br>Added chapter to install as a system service |
| 06/21/2019 | 1.2 | Removed tcl related command<br>Updated example of script extended diag service |
| 05/09/2019 | 1.1 | Updated picture of architecture<br>Added notes for unit of mem size for mem_dmi<br>Added error code for pcie_link |
| 01/29/2019 | 1.0 | Official release |
| 10/17/2018 | 0.8 | Added IPv6 setting |
| 10/11/2018 | 0.7 | Updated Install chapter |
| 10/08/2018 | 0.6 | Fixed typo |
| 08/28/2018 | 0.5 | Renamed |
| 08/21/2018 | 0.4 | Updated install steps<br>Added chapter for diagnostic functions |
| 04/25/2018 | 0.3 | Added new command loads |
| 04/08/2018 | 0.2 | Added new command loadp |
| 03/05/2018 | 0.1 | Initial version |

# We Appreciate Your Input

Please let us know of any issues with this product, including the manual, which could benefit from improvements or corrections. We appreciate your valuable input in helping make our products better.

Please send all such - in writing to: tse.ncg@advantech.com

# Table of Contents

# List of Figures



# List of Tables

# 1. INTRODUCTION

## 1.1 Overview

This document is the user manual for Advantech Diagnostic Framework software. Diagnostic Framework is a hardware diagnostic software framework. It provides some diagnostic services and some plugins to offer outside access. With the plugins, we can list all the diagnostic functions and run them. Diagnostic Framework runs on Linux OS, and supports different HW platforms.

Diagnostic Framework Architecture:

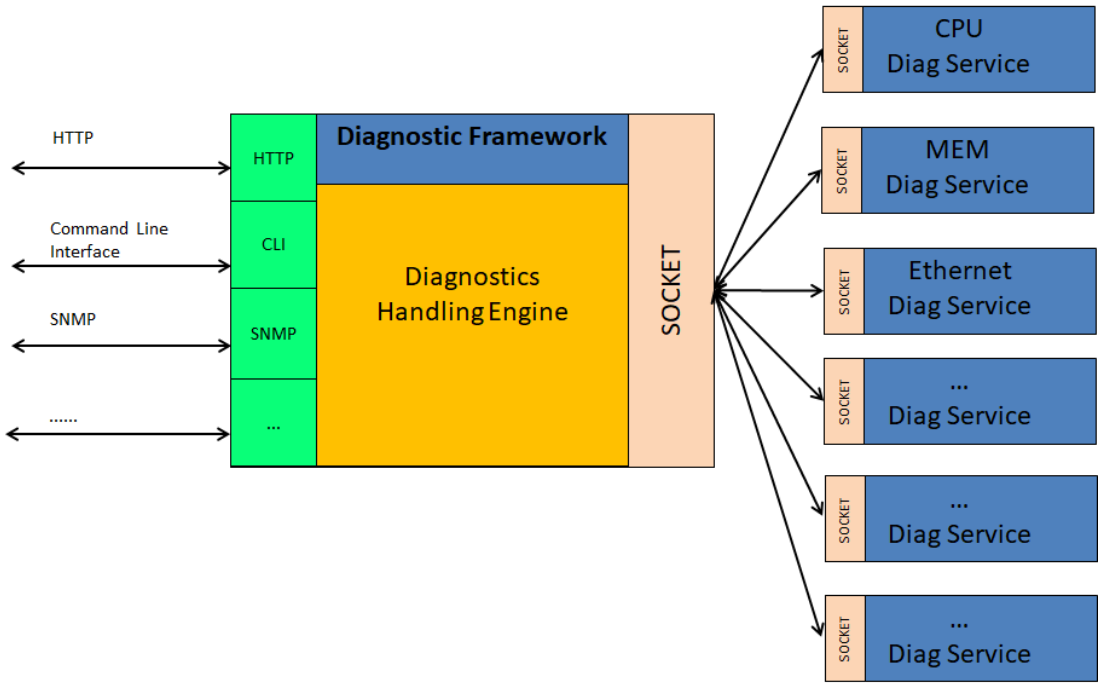



**Figure 1: Diagnostic Framework Block Diagram**

## 1.2 Requirements

Diagnostic Framework is able to run on x86 platforms and Linux OS with some library or software installed. These are the libraries required by Diagnostic Framework:

| Library | Description |
|---|---|
| libxml2 | A library to read XML configuration file |
| net-snmp | (Optional) A library to support SNMP interface, only needed when SNMP plugin is enabled. |
| tcl | (Optional) A library to support Tcl script in Diagnostic Framework CLI |

**Table 1: Library**

There are also some third-party utilities needed by diagnostic services of Diagnostic Framework:

| Software | Description |
|---|---|
| ipmitool | A utility to send IPMI commands to BMC, can be installed with package manager of the Linux distribution. |
| ethtool | A utility to test Ethernet interface, can be installed with package manager of the Linux distribution. |
| stresscpu2 | A utility to do CPU stress tests, can be downloaded from the web site: http://www.gromacs.org/Downloads/User_contributions/Other_software |
| memtester | A utility to do memory tests, can be installed with package manager of the Linux distribution. |

**Table 2: Third-party software**

## 1.3 Terminology

| Terminology | Description |
|---|---|
| IPMI | Intelligent Platform Management Interface |
| SNMP | Simple Network Management Protocol |
| HTTP | Hypertext Transfer Protocol |
| CLI | Command Line Interface |

**Table 3: Terminology**

# 2. INSTALLATION

The Diagnostic Framework provides a binary package with an install tool for Linux systems. Before installing Diagnostic Framework, the user needs to install the required libraries and software listed in Chapter 1.2.

## 2.1 Install from binary package

**Step 1**: Unpack the binary package

```
$ tar xvf diagnostic_framework_vA.B.tgz
```

**NOTES**: A.B is the version of Diagnostic Framework, for example: diagnostic_framework_v0.64.tgz.

**Step 2**: Run the install script of Diagnostic Framework

```
$ sudo ./setup.sh
```

The Diagnostic Framework has been be installed to the /usr/local/bin/dui/ directory, and a symbolic link has been created for the Diagnostic Framework daemon automatically in /usr/bin/. Users can run Diagnostic Framework from the Linux shell:

```
$ sudo duid
```

## 2.2 Install System service

In the script/ directory of the binary package, there are two service scripts to install the Diagnostic Framework as a system service. The "duid.service" is for the system and the "duid" is for SysV init or upstart.

### 2.2.1 Systemd

Copy the service script to the systemd directory and then enable it. The Diagnostic Framework daemon will then automatically start up during Linux boot.

```
$ sudo cp duid.service /lib/systemd/system/
$ sudo systemctl enable duid.service
```

### 2.2.2 SysV init or upstart

Copy the service script to the init.d directory and then enable it. The Diagnostic Framework daemon will then automatically start up during Linux boot.

```
$ sudo cp duid.service /etc/init.d/
$ sudo chkconfig --add duid              #For Linux with SysV init.
$ sudo update-rc.d duid defaults 99   #For Linux with upstart
```

## 2.3 Uninstall system service

There is an uninstall script to remove Diagnostic Framework from the Linux system.

```
$ sudo ./uninst.sh
```

NOTES: The service script will not be removed by the uninstall script, so the user needs remove it by following these steps according to different installations in 2.2.1 Systemd or 2.2.2 SysV init or upstart.

### 2.3.1 Systemd

```
$ sudo systemctl disable duid.service
$ sudo rm /lib/systemd/system/duid.service
```

### 2.3.2 SysV init or upstart

```
$ sudo chkconfig duid off
$ sudo chkconfig --del duid              #For Linux with SysV init.

$ sudo update-rc.d  -f duid remove   #For Linux with upstart

$ sudo rm /etc/init.d/duid.service
```

# 3. USAGE

## 3.1 Start Diagnostic Framework daemon

To start the Diagnostic Framework, we need just run Linux in a command shell:

```
# duid
```

**NOTE**: You need to be a super user to run the Diagnostic Framework software!

Diagnostic Framework supports running as a Linux daemon (a process running in background).

```
# duid -d
```

The Diagnostic Framework daemon should create a PID file "/var/run/duid.pid". Users can use the pid recorded in it to send a SIGTERM signal to gracefully shutdown the Diagnostic Framework daemon.

```
# kill $(cat /var/run/duid.pid)
```

**NOTES**: When starting as a Linux daemon, the CLI plugin will not load, as it is useless while Diagnostic Framework is running in the background.

## 3.2 Use HTTP interface

Diagnostic Framework provides a HTTP interface and a client to send requests to the Diagnostic Framework daemon. It works as a HTTP server and runs diagnostic services from a HTTP client (or browser).

**NOTES**: This is the recommended interface for users to send commands to Diagnostic Framework.

### 3.2.1 dui_cmd utility

Diagnostic Framework also provides a utility "dui_cmd" to send commands via the HTTP interface. This utility is a command line tool for Diagnostic Framework. It is suitable for calling in bash or other Linux scripts. All commands are sent to Diagnostic Framework via the HTTP interface, which implemented by the HTTP plugin.

```
Usage:
dui_cmd [option] <cmd><serv_id><func_id> [param1=value1] ... [paramN=valueN]
cmd      - Diagnostic commands: start, stop, status, info, result, log, wait, ...
serv_id - Diagnostic service id(>=1)
func_id - Diagnostic function id(>=0)
paramN - Parameters of diagnostic function

option:
```

```
-a <host>    The host IP of HTTP plugin
-p <port>    The port number of HTTP plugin(0~65535)
-j           Get JSON format output
-h           Print help message
```

For example:

```
$ dui_cmd start 1 0
$ dui_cmd status 2 1
$ dui_cmd result 3 0
$ dui_cmd log 4 2
$ dui_cmd wait 1 0 timeout=10
```

The dui_cmd uses the default HTTP port **8086**. If a user changes the port number of the Diagnostic Framework HTTP plugin (in dui.xml), then they need to modify the port number in the config file.

The dui_cmd connects to the Diagnostic Framework HTTP plugin running on **localhost** (IP address: 127.0.0.1). If Diagnostic Framework runs on another host, please modify the IP address in the config file.

If the config file does not exist, then dui_cmd will use the default setting.

Following is an example of the config file "dui_cmd.cfg":

```
[http]
host=127.0.0.1
port=8086
```

Upon successful return of the diagnostic command, dui_cmd returns 0, and the output of the diagnostic command is printed to stdout. If any errors are encountered, a non-zero value is returned, and an error message is printed to stdout.

The '-j' option of dui_cmd will get a JSON format response from Diagnostic Framework.

### 3.2.2 Output format

The HTTP interface supports TEXT and JSON formats for the HTTP response. It returns a TEXT format response by default.

If the client wants a JSON response, it sends an extra header `"Accept: application/json"` to the Diagnostic Framework HTTP interface. Then Diagnostic Framework will return a JSON response with the header `"Content-Type: application/json"` to tell the client it is a JSON response.

If the Diagnostic Framework sends back a `"Content-Type: text/plain"` response, the format is TEXT.

NOTES:

(1) The client will always check the header "Content-Type" in the HTTP response, to get the format of the HTTP response.
(2) When the Diagnostic Framework HTTP interface only supports TEXT format, it may use "Content-Type: text/plain" to inform the client or just send back a no "Content-Type" in its HTTP response.
(3) To be compatible with older versions of Diagnostic Framework, if the client does not find any "Content-Type" information in the response header, the format will be TEXT by default.

The format of the request and response should look like following example.

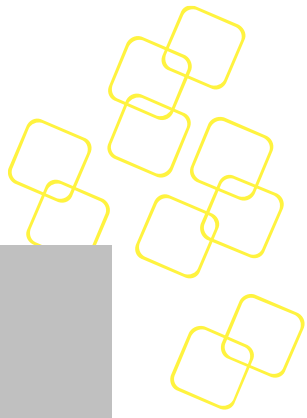
Request:

```
Get /list HTTP/1.0
Host: 127.0.0.1:8086
Accept: application/json
Connection: close
```

Response:

```
HTTP/1.0 200 OK
Server: dui_httpd/0.1
Date: Thu, 19 Sep 2019 08:19:46 GMT
Content-Type: application/json
Content-length:794
Connection: close

{
  "service": [
    {
      "id": 1,
      "name": "MEM",
      "description": "A diag service for memory",
      "function": [
        {
          "id": 0,
          "name": "mem_test",
          "description": "Memory pattern test function"
        },
        {
          "id": 1,
          "name": "mem_dmi",
          "description": "Verify memory DMI info"
        }
      ]
    },
    {
      "id": 2,
      "name": "CPU",
      "description": "A diag service for CPU",
      "function": [
```



```
      {
        "id": 0,
        "name": "cpu_test",
        "description": "CPU burn-in test function"
      },
      {
        "id": 1,
        "name": "cpu_ver",
        "description": "Verify CPU vendor id, model name, frequency and core count"
      }
    ]
  }
 ]
}
```

### 3.2.3 API

The HTTP plugin will provide the following APIs.

#### 3.2.3.1 list

This API lists all diagnostic services and functions.

The syntax of the API is:

```
"server_address:http_port/list"
```

**NOTES**:

1. The server address is the IP address of the HTTP server; we will assume it is 127.0.0.1 in the following examples.
2. The http port is the http server port number defined in the dui.xml.

Sends commands via a browser:

```
http://127.0.0.1:8086/list
```

Sends commands via dui_cmd:

```
$ dui_cmd list
```

#### 3.2.3.2 info

This API returns the information of the diagnostic function: parameters …

The syntax of the API is:

```
"server_address:http_port/info?serv_id=x&func_id=y"
```

**NOTES**:

1. The serv_id (service id) must be the first parameter.
2. The func_id (function id) must be the second parameter.

Sends commands via a browser:

```
http://127.0.0.1:8086/info?serv_id=1&func_id=2
```

Sends commands via dui_cmd:

```
$ dui_cmd info 1 2
```

#### 3.2.3.3 start

This API starts a diagnostic function. We can use serv_id and func_id to specify which functions to run.

The syntax of the API is:

```
"server_address:http_port/start?serv_id=x&func_id=y&param1=val1&param2=val2&...&paramN=valN"
```

Sends commands via a browser:

```
http://127.0.0.1:8086/start?serv_id=2&func_id=1&param1=val1&param2=val2
```

Sends commands via dui_cmd:

```
$ dui_cmd start 2 1 param1=val1 param2=val2
```

#### 3.2.3.4 status

This API returns the status of the diagnostic function.

The syntax of the API is:

```
"server_address:http_port/status?serv_id=x&func_id=y"
```

Sends commands via a browser:

```
http://127.0.0.1:8086/status?serv_id=1&func_id=2
```

Sends commands by dui_cmd:

```
$ dui_cmd status 1 2
```

#### 3.2.3.5 result

This API returns the result of the last run diagnostic function.

The syntax of the API is:

```
"server_address:http_port/result?serv_id=x&func_id=y"
```

Sends commands via a browser:

```
http://127.0.0.1:8086/result?serv_id=1&func_id=2
```

Sends commands via dui_cmd:

```
$ dui_cmd result 1 2
```

#### 3.2.3.6 stop

This API cancels the running of the diagnostic function.

The syntax of the API is:

```
"server_address:http_port/stop?serv_id=x&func_id=y"
```

Sends commands via a browser:

```
http://127.0.0.1:8086/stop?serv_id=1&func_id=2
```

Sends commands via dui_cmd:

```
$ dui_cmd stop 1 2
```

#### 3.2.3.7 log

This API returns the log message of the last run diagnostic function.

The syntax of the API is:

```
"server_address:http_port/log?serv_id=x&func_id=y"
```

Sends command via a browser:

```
http://127.0.0.1:8086/log?serv_id=1&func_id=2
```

Sends commands via dui_cmd:

```
$ dui_cmd log 1 2
```

#### 3.2.3.8 wait

This API waits for a running diagnostic function to be completed.

The syntax of the API is:

```
"server_address:http_port/wait?serv_id=x&func_id=y"
"server_address:http_port/wait?serv_id=x&func_id=y&timeout=n"
```

Sends commands via a browser:

```
http://127.0.0.1:8086/wait?serv_id=1&func_id=2
http://127.0.0.1:8086/wait?serv_id=1&func_id=2&timeout=3
```

Sends commands via dui_cmd:

```
$ dui_cmd wait 1 2
$ dui_cmd wait 1 2 timeout=3
```

#### 3.2.3.9 set

This API sets the current value of parameters of a diagnostic function.

The syntax of the API is:

```
"server_address:http_port/set?serv_id=x&func_id=y&param1=val1 "
```

Sends commands via a browser:

```
http://127.0.0.1:8086/set?serv_id=2&func_id=1&param1=val1
```

Sends commands via dui_cmd:

```
$ dui_cmd set 2 1 param1=val1
```

#### 3.2.3.10 quit

This API exits the Diagnostic Framework daemon.

The syntax of the API is:

```
"server_address:http_port/quit"
```

Sends commands via a browser:

```
http://127.0.0.1:8086/quit
```

Sends command via dui_cmd:

```
$ dui_cmd quit
```

#### 3.2.3.11 loadp

This API loads a new northbound plugin to the Diagnostic Framework daemon.

The syntax of the API is:

```
"server_address:http_port/loadp?plugin_name"
```

Sends commands via browser:

```
http://127.0.0.1:8086/loadp?cli
```

Sends command via dui_cmd:

```
$ dui_cmd loadp cli
```

#### 3.2.3.12 loads

This API loads a new diagnostic service to the Diagnostic Framework daemon.

The syntax of the API is:

```
"server_address:http_port/loads?service_uid"
```

Sends commands via a browser:

```
http://127.0.0.1:8086/loads?17
```

Sends commands via dui_cmd:

```
$ dui_cmd loads 17
```

## 3.3 Use CLI interface

Diagnostic Framework provides a CLI (command-line interpreter) to run the diagnostic services.

The CLI interface provides a command shell and will print a prompt string and wait for input from the user. CLI supports these features:

- A command line edit engine that supports a history list.
- A command line edit engine that supports the following special keys:
  - Up Arrow Key: Moves backward through the command history
  - Down Arrow Key: Moves forward through the command history.
  - Left Arrow key: Moves the cursor one character to the left on the command line.
  - Right Arrow key: Moves the cursor one character to the right on the command line.
  - Backspace key: Deletes one character on the left of the cursor.
  - Delete key: Deletes one character on the right of the cursor.

**NOTE**: This interface has a limitation. The max length of the command line is limited by the width of terminal/console.

The CLI provides some commands for the user to manage diagnostic functions.

### 3.3.1 list

This command lists all diagnostic services and functions.

The syntax of the command is:

```
cli> list
```

For example:

```
cli> list
service 1: MEM, A diag service for memory
        function 0: mem_test, Memory pattern test function
        function 1: mem_dmi, Verify memory DMI info
service 2: CPU, A diag service for cpu
        function 0: cpu_test, CPU burn in test function
        function 1: cpu_ver, Verify CPU vendor id, model name, frequency and core count
......
```

### 3.3.2 info

This command shows info about the diagnostic function: parameters.

The syntax of the command is:

```
cli> info <service id> <function id>
```

For example:

```
cli> info 1 1
```

```
Information of service 1 function 1:
Name: mem_dmi
Description: Verify memory DMI info
Parameters:
        mem_amount, Memory count
                param type   : pt_uint32
                default value: 8
                current value: 8
        mem_size, Memory size
                param type   : pt_uint32
                default value: 8192
                current value: 8192
        mem_freq, Memory frequency
                param type   : pt_uint32
                default value: 1600
                current value: 1600
```

### 3.3.3 start

Using this command, we can start a diagnostic function.

The syntax of the command is:

```
cli> start <service id> [function id] [parameter=value]
```

For example:

```
cli> start 1 0
start the function 0 of service 1


cli> start 1 0 amount=100
start service...
```

We can also start all functions of a diagnostic service by using the "start" command without function id:

```
cli> start 1
```

### 3.3.4 status

Using this command, we can get the status of the diagnostic function.

The syntax of the command is:

```
cli> status <service id> <function id>
```

For example:

```
cli> status 1 1
status code: st_ready
```

### 3.3.5 result

Using this command, we can get the results of last running of a diagnostic function.

The syntax of the command is:

```
cli> result <service id> <function id>
```

For example:

```
cli> result 1 0
exit code    : ERR_UNEXPECTED_VALUE
error message: Unexpected value.
success count: 0
fail count   : 1
test log     : /var/log/dui/dui_01_00_20120306_164120.log
```

### 3.3.6 stop

Using this command, we can stop a running diagnostic function.

The syntax of the command is:

```
cli> stop <service id> <function id>
```

For example:

```
cli> stop 1 0
service stopped.
```

### 3.3.7 log

Using this command, we can dump the log of a diagnostic function.

The syntax of the command is:

```
cli> log <service id> <function id>
```

For example:

```
cli> log 1 1
| 14:04:39 01/05/2015 | [INFO] | service name: CPU
| 14:04:39 01/05/2015 | [INFO] | function name: cpu_test
| 14:04:39 01/05/2015 | [INFO] | duration: 100
| 14:04:39 01/05/2015 | [INFO] | Running command "stresscpu2"
| 14:04:39 01/05/2015 | [INFO] | ...
| 14:07:59 01/05/2015 | [RESULT] | [PASS]
```

### 3.3.8 wait

Using this command, we can wait for a running diagnostic function to be completed.

The syntax of the command is:

```
cli> wait <service id> <function id> [timeout=n]
```

For example:

```
cli> wait 1 0
wait function completed.
wait function OK.
```

We can also specify a timeout value to make sure the wait command will return after some seconds.

```
cli> wait 1 0 timeout=5
wait function completed.
wait function timeout.
```

### 3.3.9 set

Using this command, we can set the current parameter values of a diagnostic function.

The syntax of the command is:

```
cli> set <service id> <function id> <parameter=value>
```

For example:

```
cli> set 1 0 amount=100
set function 0's parameter for the service 1.
set function OK.
```

### 3.3.10 quit

This command exits the Diagnostic Framework daemon.

The syntax of the command is:

```
cli> quit
```

### 3.3.11 loadp

This command loads a new northbound plugin to the Diagnostic Framework daemon.

The syntax of the command is:

```
cli> loadp <plugin_name>
```

For example:

```
cli> loadp http
```

### 3.3.12 loads

This command loads a new diagnostic service to the Diagnostic Framework daemon.

The syntax of the command is:

```
cli> loads <service_uid>
```

For example:

```
cli> loads 17
```

### 3.3.13 help

This CLI command shows help info for CLI commands.

For example:

```
cli> help start
Start a diagnostic function
Usage: start <service id> [function id] [param_name=param_value]
```

The "help" command without an argument lists all available commands:

```
cli> help
Available commands list:
quit    list    start    stop    status
result  info    run      set     log
wait
```

## 3.4 Configuration

### 3.4.1 dui.xml

This Diagnostic Framework software uses an XML configuration file to configure all diagnostic services and plugins. The name of the config file is "dui.xml" by default.

Diagnostic Framework loads all diagnostic services and plugins in the "dui.xml" during startup.

```
<?xml version="1.0" encoding="UTF-8" ?>
<dui>
<southbound>
  <service name="CPU">
    <uid>00000001</uid>
  </service>

  <service name="MEM">
    <uid>00000002</uid>
  </service>

  ......

</southbound>

<northbound>
  <plugin name="CLI">
    <path>libcli.so</path>
  </plugin>

  <plugin name="HTTP">
    <path>libhttp.so</path>
    <port>8086</port>
  </plugin>

  ......

</northbound>
```

```
</dui>
```

**NOTES**:

- The "<uid>" tag defines the filename of the diagnostic service.
- The "<path>" tag defines the filename of the plugin.
- The parameters of plugins are defined in the section on the plugin. For example, there is a "<port>" tag that defines the port for the HTTP interface.
- After you have modified the config file, please restart the Diagnostic Framework for the daemon to take effect.

The HTTP interface supports IPv6, and is disabled by default. It can be enabled by adding an option to dui.xml as follows.

```
<plugin name="HTTP">
  <path>libhttp.so</path>
  <port>8086</port>
  <ipv6>true</ipv6>
</plugin>
```

### 3.4.2 dui_serv.cfg

The default values of the parameters of the diagnostic service can be set in the config file. All diagnostic services will load their default parameter values from the config file when the service starts.

The default filename of the config file is dui_serv.cfg. All services share one config file. The syntax of the config file is an enhanced format of INI:

```
[“service name”-“function name”]
parameter1=value1  ;this is comment
parameter2=value2
```

For example:

```
[MEM-mem_dmi]
mem_amount=2
mem_size=2048
mem_freq=1066

[MEM-mem_test]
amount=10

[CPU-cpu_test]
duration=10
```

```
[CPU-cpu_ver]
cpu_id=GenuineIntel
cpu_model=L5815
frequency=1.40
core_count=32
……
```

### 3.4.3 dui_cfg

This is a utility to detect the HW information (include CPU, MEM, PCI, and Storage devices) and create a dui_serv.cfg for the HW platform automatically.

To run the dui_cfg utility under Linux shell:

```
$ sudo dui_cfg
Config file generated: dui_serv.cfg
```

A configuration file is created with the right parameters for this HW platform, and can be used to run Diagnostic Framework on this HW platform.

NOTES:

(1) After coping the dui_serv.cfg to directory of Diagnostic Framework, please restart the duid.
(2) Please remove the config file tui.json to make the TUID synchronize all parameters from duid.

## 3.5 Log

The output of diagnostic will be saved in a log file. All log files will be placed under /var/log/dui/ directory, and the filename of the log file should follow the rule:

```
dui_<Serv_ID>_<Func_ID>_<Date_Time>.log
```

- Serv_ID : The service id
- Func_ID : The function id
- Date_Time : Date and time of the log file created.

For example, the log file of service 2 function 1 should be:

```
/var/log/dui/dui_02_01_20111205_150928001.log
```

The format of the log message should be in the following style:

```
| DATE_TIME | [SEVERITY] | Event
```

- DATE_TIME: The date and time of the message generated.
- SEVERITY: INFO/WARNING/ERROR/RESULT/DEBUG.
  - INFO: Provides some information.
  - WARNING: A warning.
  - ERROR: An error occurred.
  - RESULT: The result of the diagnostic function.
  - DEBUG: Debug message.
- Event: The log message.

For example:

```
| 14:04:39 01/05/2015 | [INFO] | service name: CPU
| 14:04:39 01/05/2015 | [INFO] | function name: cpu_test
| 14:04:39 01/05/2015 | [INFO] | duration : 100
| 14:04:39 01/05/2015 | [INFO] | Running command "stresscpu2"
| 14:04:39 01/05/2015 | [INFO] | The result of command "stresscpu2" is saved in file "/var/log/dui/dui_stresscpu2_01_00_20150105_140439066.log"
| 14:07:59 01/05/2015 | [RESULT] | [PASS]…
```

The output of the third party tool or command used in diagnostic functions will also be saved in another log file. These log files will be placed under the same directory /var/log/dui/. The filename of these log files will have the same style as the above log files and include the name of the third party tools in it.

For example, the log file of ipmitool used in service 3 function 0 should be:

```
/var/log/dui/dui_ipmitool_03_00_20111217_162541001.log
```

# 4. DIAGNOSTIC SERVICE

## 4.1 Service id and Function id

Each service will have a service id, and the service id will be unique and equal to the UID.

Each function will have a function id, and the function id will be unique to the diagnostic service.

## 4.2 Service list

### 4.2.1 Overview

Diagnostic Framework provides some services and functions, some of them are enabled by default:

| Service | Function | Description | Depends |
|---|---|---|---|
| CPU | cpu_test | CPU Burn in test | stresscpu2 |
| | cpu_ver | Verify CPU vendor id, model, frequency and core count | |
| MEM | mem_test | Runs memory test | memtester |
| | mem_dmi | Validates DMI RAM information | |
| LAN | lan_func | Does internal loopback test on LAN interfaces | ethtool |
| | lan_mac | Duplicated MAC address detection | |
| Storage | ssd_smart | SATA/SAS/SSD Disk SMART test | smartctl |
| | sector_check | Check the sector of SATA/SAS/SSD disk. | badblocks |
| | benchmark | Storage benchmark | dd |
| RTC | rtc_sys | System RTC test | |
| PCI | pcie_link | Verifies PCI link width and status | lspci |
| | pcie_status | Check error status of PCIE device | lspci |

**Table 4: Service and function enabled by default**

There are some services/functions disabled by default:

| Service | Function | Description | Depends |
|---|---|---|---|
| IPMC | ipmc_eep | Verifies EEPROM access | ipmitool |
| | ipmc_sensor | Verifies sensors | ipmitool |
| | ipmc_i2c | IPMC Master Only I2C test | ipmitool |
| | ipmc_ipmb0 | Verifies the IPMB0 LINK | ipmitool |
| | ipmc_fru | Verifies IPMC FRU data | ipmitool |
| | ncsi_test | NCSI interface test | ipmitool |
| | ipmc_ver | IPMC Firmware version check | ipmitool |
| | rtm_mmc_ver | MMC version check | ipmitool |
| Script | | Runs Linux shell script | |

**Table 5: Services and functions disabled by default**

If you want to enable these services or functions, please modify the configuration file dui.xml. There is a template file dui.xml.full; users can copy contents from it to dui.xml. There is also a template file dui_serv.cfg.full for default parameters of the diagnostic functions.

For example, if we need to enable a script service, please copy the following content in RED text into the <southbound> section of dui.xml:

```
<southbound>

  <service name="SCRIPT">
     <uid>00000007</uid>
   </service>

</southbound>
```

### 4.2.2 CPU

#### 4.2.2.1 cpu_test - CPU Burn in test

This diagnostic function runs the CPU burn-in utility (stresscpu2) for a specified duration to test the CPU.

PARAMETERS:

==========

duration - Duration of the test (in seconds)

RETURN VALUES:

==========

ERR_SUCCESS - Burn-in utility has been running for a specified time.

ERR_CPU_BURNIN_USING - Error on invoking burn-in utility

ERR_INVALID_PARAMETER - Invalid parameter

#### 4.2.2.2 cpu_ver - Verify CPU vendor id, model, frequency and core count

Diagnostic function Parse /proc/cpuinfo verifies CPU vendor id, model, frequency and core count.

PARAMETERS:

==========

cpu_id - Vendor ID (For Intel CPU, it should be “Genuine Intel“)

cpu_model - Model Name

frequency - Frequency (in GHz)

core_count - CPU core count

RETURN VALUES:

==========

ERR_SUCCESS - All criteria match.

ERR_PROC_CPUINFO_ACCESS - /proc/cpuinfo cannot be accessed.

ERR_CPU_INV_VENDOR - The vendor string does not match

ERR_CPU_INV_MODEL - The model name does not match

ERR_CPU_INV_FREQ - The frequency does not match

ERR_CPU_INV_COUNT - The core count does not match

ERR_INVALID_PARAMETER - Invalid parameter

### 4.2.3 MEM

#### 4.2.3.1 mem_test - Run memory test

This diagnostic function uses memtester to test the memory.

PARAMETERS:

==========

amount - Size of RAM to test (in MB/GB, If no UNIT specified, it's MB)
If amount > 0, run memtester in single process.
If amount = 0, then this diagnostic function will run multiple memtester processes in parallel. It will detect the total free RAM in the system and decide how much RAM needs to be tested.

RETURN VALUES:

==========

ERR_SUCCESS - Memory test completed successfully
ERR_COMMAND_NOT_FOUND - Command is not found
ERR_RAM_NOT_ENOUGH - There is not enough free RAM to run the test
ERR_INVALID_PARAMETER - Invalid parameter
ERR_RAM_MEMTEST_FAIL - Memory test failed

#### 4.2.3.2 mem_dmi - Validate DMI RAM information

This diagnostic function retrieves the following information from BIOS DMI:

- Amount of memory modules
- Size of memory modules
- Frequency of memory modules

PARAMETERS:

==========

mem_amount - Amount of DIMMs
mem_size - Size of each DIMM (in MB/GB. If no UNIT is specified, it's MB)
mem_freq - Frequency of each DIMM (in MHz)

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match. |
| ERR_RAM_AMOUNT_MISMATCH | - Amount of DIMMs does not match |
| ERR_RAM_DIMM_SIZE_MISMATCH | - Size of DIMM does not match |
| ERR_RAM_DIMM_FREQ_MISMATCH | - Frequency of DIMM does not match |
| ERR_OPEN_FILE_ERR | - Opening sysfs failed |
| ERR_READ_FILE | - Reading DMI data failed |
| ERR_MALLOC | - Allocating memory failed |
| ERR_GENERIC_FAILURE | - Other errors |

### 4.2.4 LAN

#### 4.2.4.1 lan_func – Do internal loopback test on LAN interfaces

This diagnostic function uses ethtool to do the internal loopback test on LAN interfaces.

PARAMETERS:

==========

bus - PCIe bus id of the checked device, support multiple devices: "0000:01:00.0,0000:02:00.0"

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - Loopback tested successfully |
| ERR_OPEN_FILE_ERR | - Open file failed |
| ERR_RUN_SYSTEM_CMD | - Running system command failed |
| ERR_LAN_LOOPBACK_FAIL | - Ethernet loopback test failed |

#### 4.2.4.2 lan_mac – Duplicated MAC address detection

This diagnostic function checks if there are duplicated MAC addresses of all LAN controllers and IPMC.

PARAMETERS:

==========

No associated parameters.

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - No duplicated MAC address found. |
| ERR_LAN_MAC_DUPLICATE | - Duplicated MAC address found. |
| ERR_RUN_SYSTEM_CMD | - Running system command failed. |

#### 4.2.4.3 lan_stat – Checks the statistics of a network device

This diagnostic function checks the statistics of a network device via /sys/class/net/ETH/statistic.

PARAMETERS:

==========

dev_list - device list (e.g.: eth0, eth1, eth2; all_dev for all device)

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - Loopback tested successfully. |
| ERR_OPEN_FILE_ERR | - Open file failed. |
| ERR_READ_FILE | - Read file failed. |
| ERR_UNEXPECTED_VALUE | - Unexpected value. |
| ERR_ETH_DEVICE_MISSING | - The network device cannot be found. |

### 4.2.5 Storage

#### 4.2.5.1 ssd_smart – SATA/SAS/SSD SMART test

Diagnostic function SSD/SATA/SAS SMART tests memory via smartctl utility.

PARAMETERS:

==========

dev_list - Storage device list (e.g.: sda, sdb, sdc, …)

RETURN VALUES:

==========

ERR_SUCCESS - SMART check OK

ERR_CMD_NOT_FOUND - Cannot find the command/utility

ERR_SMART_FAIL - SMART test fail

#### 4.2.5.2 sector_check – Check the sector of disk.

This diagnostic function checks the sectors of DISK (SATA/SAS/SSD) via bad blocks.

PARAMETERS:

==========

dev_list - Storage device list (e.g.: sda, sdb, sdc, …)

RETURN VALUES:

==========

ERR_SUCCESS - Sector test OK

ERR_CMD_NOT_FOUND - Cannot find the command/utility

ERR_SECTOR_FAIL - Sector test fail

#### 4.2.5.3 benchmark - Storage benchmark

Diagnostic function does a read/write test on a specified storage device (hard disk, USB flash drive), and computes the speed of the read/write.

PARAMETERS:

==========

dev_list - The storage device list separated with comma, e.g.: sda, sdb, sdc, …

RETURN VALUES:

==========

ERR_SUCCESS - OK

ERR_RUN_SYSTEM_CMD - Running system command failed

### 4.2.6 RTC

#### 4.2.6.1 rtc_sys – System RTC test

This diagnostic function tests the System RTC.

PARAMETERS:

==========

No associated parameters.

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match. |
| ERR_RTC_SYSTEM_READ | - Error on reading RTC time |
| ERR_RTC_SYSTEM_WRITE | - Error on writing RTC time |
| ERR_RTC_SYSTEM_RESTORE | - Error on restoring RTC time |
| ERR_RTC_SYSTEM_COMPARE | - The read back RTC time is out of range |
| ERR_OPEN_FILE_ERR | - Open selected file failed |

### 4.2.7 PCI

#### 4.2.7.1 pcie_link - Verify PCI link width and status

This diagnostic function reads the PCI-Express root port link status register to determine the link state of external PCI Express devices.

PARAMETERS:

==========

| | |
|---|---|
| addr | - PCI bus id of PCI Express device: 0000:01:00.0 |
| width | - PCI Express link width: x8 … |
| speed | - PCI Express link speed: 5G … |
| train | - PCI Express link training: 1 - active, 0 - not active. |

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - If PCIe link width and status are as expected |
| ERR_PCI_CS_ACCESS_FAIL | - If /proc/bus/pci/x:x.x (config space) cannot be accessed |
| ERR_UNEXPECTED_VALUE | - If register content is wrong |

ERR_RUN_SYSTEM_CMD - Running system command failed

ERR_STRING_NOT_FOUND - There is no "LnkSta" in the output of lspci

#### 4.2.7.2 pcie_status - Check error status of PCIE device

This diagnostic function reads the PCI-Express device status register to check the fatal, non-fatal and correctable errors of PCI Express devices.

PARAMETERS:

==========

bus - Bus number of the checked device (supports multiple devices: "0000:01:00.0, 0000:02:00.0")

RETURN VALUES:

==========

ERR_SUCCESS - If no errors found

ERR_PCIE_STATUS_ERROR - Fatal/non-fatal error found

ERR_INVALID_PARAMETER - Parse parameters fail

ERR_UNEXPECTED_VALUE - No DevSta: found in output of lspci

ERR_RUN_SYSTEM_CMD - Running system command failed

### 4.2.8 IPMC

All the diagnostic functions belong to this service only workable on hardware platforms with Advantech IPMI solution.

#### 4.2.8.1 ipmc_eep– Verify EEPROM access

This diagnostic function uses the ipmitool to write some patterns to a reserved area in the FRU EEPROM, then reads it back and compares.

PARAMETERS:

==========

No associated parameters.

RETURN VALUES:

==========

ERR_SUCCESS - All criteria match.

| | |
|---|---|
| ERR_RUN_SYSTEM_CMD | - Running system command failed |
| ERR_OPEN_FILE_ERR | - The temporary file of ipmitool is not readable |
| ERR_FRU_EEPROM_WRITE | - Write access to EEPROM failed |
| ERR_FRU_EEPROM_READ | - The data read back fails |

#### 4.2.8.2 ipmc_sensor - Verify sensors

This diagnostic function uses the ipmitool to read all sensors and check that no event is asserted for the voltage and temperature sensors.

PARAMETERS:

==========

| | |
|---|---|
| sdr_file | - The SDR file generated by “ipmitool sdr” command. |

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match. |
| ERR_RUN_SYSTEM_CMD | - Running system command failed |
| ERR_OPEN_FILE_ERR | - The temporary file of ipmitool is not readable |
| ERR_STRING_OP_ERROR | - A string operation fails |
| ERR_SENSOR_MISSING | - SDR record missing |
| ERR_SENSOR_FAIL | - SDR record reading failed |

#### 4.2.8.3 ipmc_i2c - IPMC Master Only I2C test

This diagnostic function uses the ipmitool to access the sensor on the master only I2C bus. It then checks for device accessibility (ACK on I2C bus) as well as expected register contents.

PARAMETERS:

==========

No associated parameters.

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match. |
| ERR_RUN_SYSTEM_CMD | - Running system command failed |

| | |
|---|---|
| ERR_OPEN_FILE_ERR | - The temporary file of ipmitool is not readable |
| ERR_MONLY_I2C_NACK | - There is no acknowledge on the I2C bus |
| ERR_GENERIC_FAILURE | - Other errors |

#### 4.2.8.4 ipmc_ipmb0 - Verify the IPMB0 LINK

This diagnostic function uses the ipmitool to do an RTC sync with Shelf Manager and evaluates the response. The response from Shelf Manager will confirm the health of the IPMB0 link.

PARAMETERS:

==========

No associated parameters.

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match. |
| ERR_RUN_SYSTEM_CMD | - Running system command failed |
| ERR_OPEN_FILE_ERR | - The temporary file of ipmitool is not readable |
| ERR_IPMB0_FAIL | - The communication does not work correct |
| ERR_IPMB0_RTC_SYNC_GET | - Get IPMI RTC sync setting error |
| ERR_IPMB0_RTC_SYNC_SET | - Change IPMI RTC sync setting error |

#### 4.2.8.5 ipmc_fru – Verify IPMC FRU data

This diagnostic function reads the FRU data from the EEPROM and verifies the checksum and static data.

PARAMETERS:

==========

| | |
|---|---|
| fru_number | - Attached FRU device number |
| manufacturer | - The board manufacturer, e.g. Advantech |
| product_name | - The product name, e.g. MIC-5344 |
| hw_version | - The board HW version, e.g. A103 |

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match |
| ERR_RUN_SYSTEM_CMD | - Run system command failed |
| ERR_OPEN_FILE_ERR | - The temporary file of ipmitool is not readable |
| ERR_READ_FRU | - The FRU data is not same as default |

#### 4.2.8.6 ncsi_test – NCSI interface test

This diagnostic function tests the NCSI interface.

PARAMETERS:

==========

No associated parameters.

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match |
| ERR_RUN_SYSTEM_CMD | - Running system command failed |
| ERR_INVALID_PARAMETER | - Invalid parameter |
| ERR_OPEN_FILE_ERR | - Open file failed |
| ERR_INVALID_IPADDRESS | - Invalid IP address |

#### 4.2.8.7 ipmc_ver– IPMC version check

This diagnostic function checks the IPMC version (via ipmitool).

PARAMETERS:

==========

| | |
|---|---|
| hpm_version | - The IPMC firmware version (HPM.1) |

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match |
| ERR_RUN_SYSTEM_CMD | - Run system command failed |
| ERR_STRING_OP_ERROR | - Could not extract IPMC version |
| ERR_IPMC_HPM_DIFFER | - Active version does not match backup version |
| ERR_STRING_NOT_FOUND | - Cannot find string |

ERR_IPMC_HPM_MISMATCH - IPMC FW version (HPM.1) mismatch

#### 4.2.8.8 rtm_mmc_ver– MMC version check

This diagnostic function checks the RTM MMC version (via ipmitool).

PARAMETERS:

==========

rtm_mmc_hpm_version- The MMC version (HPM.1)

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match. |
| ERR_RUN_SYSTEM_CMD | - Run system command failed |
| ERR_STRING_OP_ERROR | - Could not extract MMC version |
| ERR_IPMC_HPM_DIFFER | - Active version does not match backup version |
| ERR_IPMC_HPM_MISMATCH | - RTM MMC version (HPM.1) mismatch |
| ERR_STRING_NOT_FOUND | - Cannot find a specified string |

### 4.2.9 SCRIPT

#### 4.2.9.1 Overview

This diagnostic function invokes Linux shell scripts and checks for string "<pass>" tests.

This service will scan all shell scripts and create a diagnostic function for each found shell script.

NOTES: This diagnostic service is disabled by default, and we need to modify the dui.xml to enable it before using it. Please refer chapter 4.2.1 for how to modify the dui.xml.

#### 4.2.9.2 Format of scripts

The requirement of the shell scripts:

(1) The file name of the scripts shall be "dui_nnn_*.sh", for example, dui_000_memtest.sh, dui_001_lantest.sh. The "nnn" part is the function id. The script service gets the function id from the filename of the script. If the function id is duplicated with another test script, then the later script will be ignored and skipped. The function id need not be continuous. If a specified function id does not exist, it will show up as an "Invalid Function" and its status will be "st_notready" and "no start/stop request supported!"

(2) All shell scripts shall be placed under the sub-directory "script/" of the Diagnostic Framework (default is /usr/local/bin/dui/script/). The script service supports up to 100 shell scripts (valid function id from 0 to 99).

(3) There will be an information section in the shell script file:

```
###FUNC INFO BEGIN
#Name: lan_test
#Description: Generate packet and send between 2 LAN ports
#Parameter: {head=-p}{packet_size=1024}{the packet size}
#Parameter: {head=-n}{packet_number=1000}{the packet number}
#Parameter: {port1=eth2}{the first LAN port}
#Parameter: {port2=eth5}{the second LAN port}
###FUNC INFO END
```

The function information section will begin with a "###FUNC INFO BEGIN" line, and end with a "###FUNC INFO END" line.

The function name shall be defined in the "#Name: " line.

The description of the function shall be defined in the "#Description: " line.

The parameters of the function shall be defined in the "#Parameter: " lines. Each script supports up to 20 parameters.

Let us take the "lan_test" function for an example to illustrate the parameters parsing method. This function invokes the dui_000_test1.sh , the parameters for the script are dui_000_test1.sh -p 1024 -n 1000 eth2 eth5.

There are two types of parameters:

The first type: regular type,(ex. eth2 eth5), these parameters do not need specific prefixes to be parsed by the dui_000_test1.sh. This kind of parameter follows the syntax:

```
{Parameter_Name=Parameter_Value}{Description of the parameter}.
```

For example,

```
#Parameter: {port1=eth2}{the first LAN port}
#Parameter: {port2=eth5}{the second LAN port}
```

The second type: special type,(ex. 1024 1000),these parameters need specific prefixes to be parsed by the dui_000_test1.sh, for example ,-p -n . which is defined as "head" in the script header. This kind of parameter follows the syntax:

```
{head=Head_Value}{Parameter_Name=Parameter_Value}{Description of the parameter}.
```

For example,

```
#Parameter: {head=-p}{packet_size=1024}{the packet size}
#Parameter: {head=-n}{packet_number=1000}{the packet number}
```

***Caution**! The Parameter_Name can not be "head" otherwise Diagnostic Framework can not parse the parameter.*

These "heads" are relatively meaningless to the Diagnostic Framework daemon, so while using the info command, the "heads" will not show up but will automatically be added as parameters while running the shell script.

For example,

```
$ dui_cmd info 7 0
Information of service 7 function 0:
Name: lan_test
Description: Generate packet and send between 2 LAN ports
Service Impact Level: NONDEGRADING
Parameters:
    packet_size, the packet size
            param type   : pt_string
            default value: 1024
            current value: 1024
    packet_number, the packet number
            param type   : pt_string
            default value: 1000
            current value: 1000
    port1, the first LAN port
            param type   : pt_string
            default value: eth2
            current value: eth2
    port2, the second LAN port
            param type   : pt_string
            default value: eth5
            current value: eth5
```

We can see that -p & -n are invisible in the info command.

```
$ dui_cmd start 7 0
start the function 0 of service 7 ...
start function OK.

$ dui_cmd log 7 0
| 14:12:45 01/07/2015 | [INFO] | service name: SCRIPT
| 14:12:45 01/07/2015 | [INFO] | function name: lan_test
| 14:12:45 01/07/2015 | [INFO] | running /usr/local/bin/dui//script//dui_000_test1.sh -
p 1024 -n 1000 eth2 eth5 2>&1
| 14:12:45 01/07/2015 | [INFO] | lan_test
| 14:12:45 01/07/2015 | [INFO] | <pass>
| 14:12:45 01/07/2015 | [RESULT] | [PASS]
```

We can see that -p & -n are automatically added as parameters while running shell script.

(4) Each shell script will print a message string "<pass>" when the test completes successfully. The script service will try to check the string to get the result of the diag function.

(5) When the Diagnostic Framework tries to stop the running shell script, it will send a signal SIGTERM to the shell script. Each shell script catches the signal SIGTERM and does some cleanup job.

(6) Diagnostic Framework supports the raw command mode to invoke the shell script. The user can expand or lessen parameter inputs without modifying the script header section.

For example,

```
$ dui_cmd start 7 0 raw="--help"
start the function 0 of service 7 ...
start function OK.

$ dui_cmd log 7 0
| 14:12:45 01/07/2015 | [INFO] | service name: SCRIPT
| 14:12:45 01/07/2015 | [INFO] | function name: lan_test
| 14:12:45 01/07/2015 | [INFO] | running /usr/local/bin/dui//script//dui_000_test1.sh --help 2>&1
| 14:12:45 01/07/2015 | [INFO] | lan_test
| 14:12:45 01/07/2015 | [INFO] | <pass>
| 14:12:45 01/07/2015 | [RESULT] | [PASS]
```

```
$ dui_cmd start 7 0 raw="-p 1024 -n 1000 -t 100 eth2 eth5"
start the function 0 of service 7 ...
start function OK.

$ dui_cmd log 7 0
| 14:12:45 01/07/2015 | [INFO] | service name: SCRIPT
| 14:12:45 01/07/2015 | [INFO] | function name: lan_test
| 14:12:45 01/07/2015 | [INFO] | running /usr/local/bin/dui//script//dui_000_test1.sh -p 1024 -n 1000 -t 100 eth2 eth5 2>&1
| 14:12:45 01/07/2015 | [INFO] | lan_test
| 14:12:45 01/07/2015 | [INFO] | <pass>
| 14:12:45 01/07/2015 | [RESULT] | [PASS]
```

Diagnostic Framework CLI plugin uses the space as delimiter, parameters inputs needs to include the symbol " " (double quotation marks), otherwise Diagnostic Framework can not correctly invoke the script.

#### 4.2.9.3 Example of script

Following is an example of the script:

```
#!/bin/sh

###FUNC INFO BEGIN
#Name: lan_test
#Description: Generate packet and send between 2 LAN ports
#Parameter: {head=-p}{packet_size=1024}{the packet size}
#Parameter: {head=-n}{packet_number=1000}{the packet number}
#Parameter: {port1=eth2}{the first LAN port}
#Parameter: {port2=eth5}{the second LAN port}
###FUNC INFO END

echo "lan_test"
```

```
term_exit()
{
   # Function to perform exit if termination signal is trapped
   echo "script terminated!"
   exit 1
}
trap term_exit TERM

#
# Do the test ...
#

#diag function completed successfully.
echo "<pass>"
exit 0
```

After Diagnostic Framework starts up, we can use these CLI commands to get information: start, get results, and the log of this function:

```
$ dui_cmd info 7 0
Information of service 7 function 0:
Name: lan_test
Description: Generate packet and send between 2 LAN ports
Service Impact Level: NONDEGRADING
Parameters:
    packet_size, the packet size
                param type   : pt_string
                default value: 1024
                current value: 1024
    packet_number, the packet number
                param type   : pt_string
                default value: 1000
                current value: 1000
    port1, the first LAN port
                param type   : pt_string
                default value: eth2
                current value: eth2
    port2, the second LAN port
                param type   : pt_string
                default value: eth5
                current value: eth5
```

```
$ dui_cmd start 7 0
start the function 0 of service 7 ...
start function OK.
```

```
$ dui_cmd result 7 0
exit code    : ERR_SUCCESS
```

```
error message: OK
success count: 1
fail count   : 0
test log     : /var/log/dui/dui_7_00_20150107_141245106.log
```

```
$ dui_cmd log 7 0
| 14:12:45 01/07/2015 | [INFO] | service name: SCRIPT
| 14:12:45 01/07/2015 | [INFO] | function name: lan_test
| 14:12:45 01/07/2015 | [INFO] | running /usr/local/bin/dui//script//dui_000_test1.sh -
p 1024 -n 1000 eth2 eth5 2>&1
| 14:12:45 01/07/2015 | [INFO] | lan_test
| 14:12:45 01/07/2015 | [INFO] | <pass>
| 14:12:45 01/07/2015 | [RESULT] | [PASS]
```

PARAMETERS:

==========

Parameters defined in the information section of Linux shell script (bash).

RETURN VALUES:

==========

| | |
|---|---|
| ERR_SUCCESS | - All criteria match |
| ERR_STRING_NOT_FOUND | - A string is expected but not be found |
| ERR_RUN_SYSTEM_CMD | - Running system command failed |
| ERR_INVALID_PARAMETER | - Invalid parameter |
| ERR_STOPPED_BY_USER | - The function stopped by user |

# 5. TUID

Diagnostic Framework provides a Terminal-UI utility "TUID" to run diagnostic functions. TUID has some windows for different operations.

## 5.1 Main window

When the TUID starts, it shows the main window.

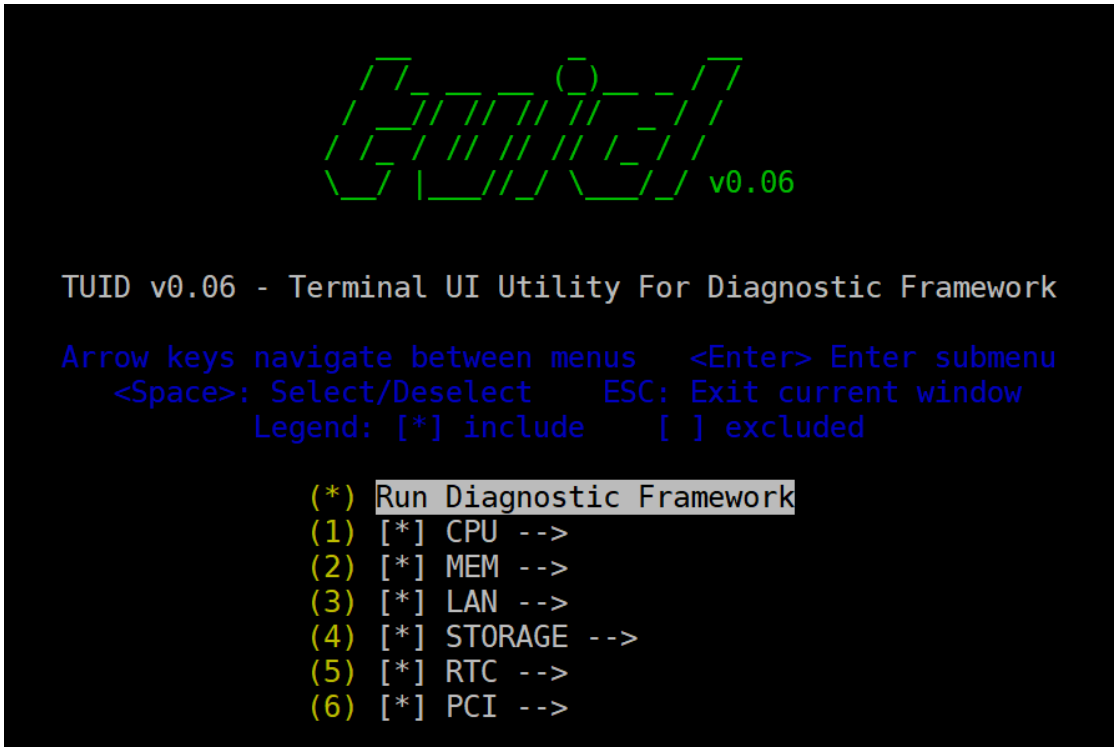



**Figure 2: Main Widows of TUID**

In this window, the user can:

- Run the Diagnostic Framework.
- Use Space key select or deselect the services, and use the Enter key to enter the submenu. Services marked with [*] will be executed.
- Use Ctrl-S or Ctrl-H key to open the Save or Help window.

## 5.2 Function window

TUID will enter the function window when the user presses the Enter key in the main window.

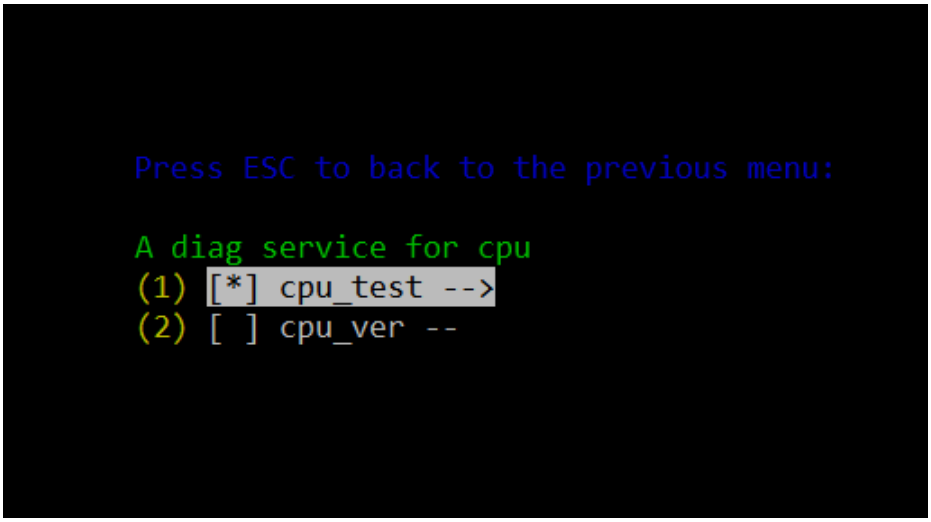



**Figure 3: Function Widows of TUID**

In this window, the user can:

- Use the ESC key to go back to the main window.
- Use the Space key to select or unselect which diagnostic functions to run. The function marked with [*] is selected.
- Enter the parameter window by pressing the Enter key.

## 5.3 Parameter window

TUID will open the parameter window when the user presses the Enter key in the function window.

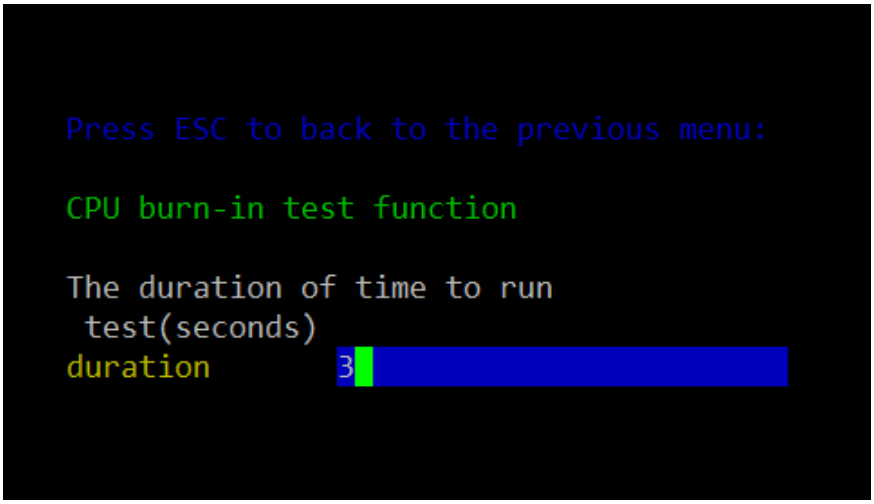



**Figure 4: Parameter Widows of TUID**

In this window, the user can:

- Use the ESC key to go back to the previous window.
- Set parameters for this function.

## 5.4 Run window

When the user presses the Enter key on the Run Diagnostic Framework item in the main window, TUID will open the run window and start running all selected diagnostic functions one by one.

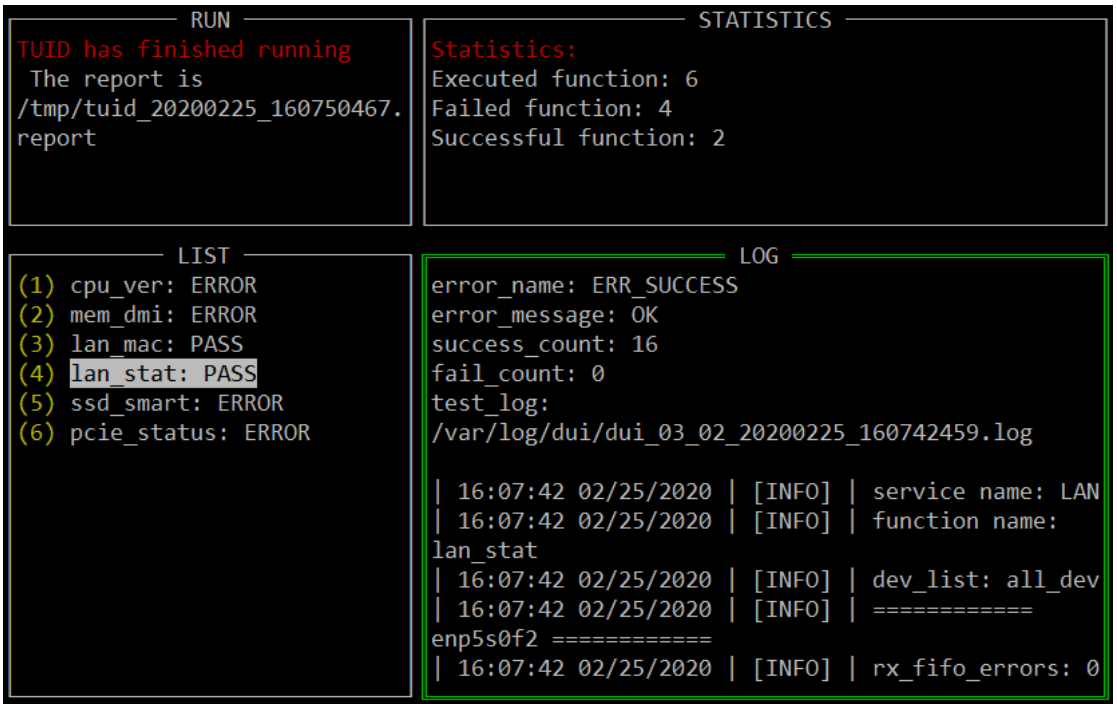



**Figure 5: Run Widows of TUID**

In this window, the user can:

- Watch the function run, and view the status of the function.
- Open the log file when pressing the Enter key of a specific function item.
- Use the ESC key to go back to the previous window.
- Use Left/Right key to switch between List sub-window and Log sub-windows.
- Use Up/Down key to select function in the List sub-window.
- Use Up/Down key to navigate to the log function in the Log sub-window.

When the user presses the Enter key on any diagnostic function in the List sub-window of run window, TUID will display the log of the function in Log sub-window.

TUID will create a report file and a log file in the /tmp/ directory. The log file will record some error message reports by TUID. The report file is a simple report of all diagnostic functions run by TUID.

## 5.5 Help window

When the user presses the CTRL-H key in the main window or the submenu window, TUID will open the help window, and display a help message from TUID.

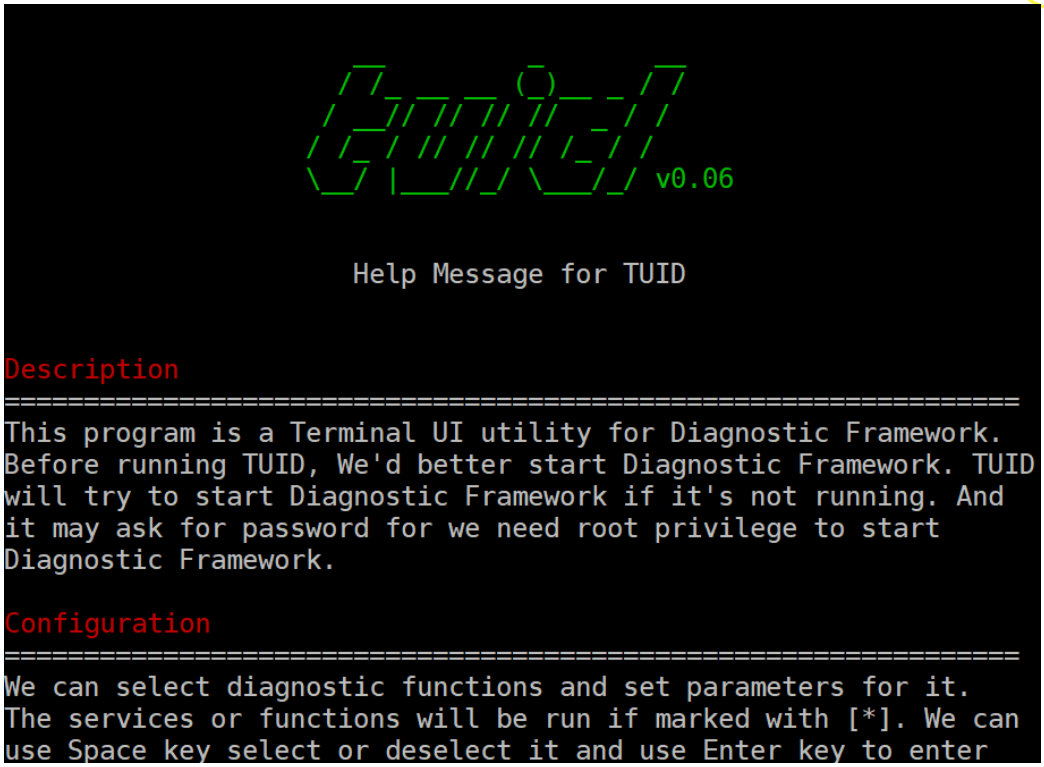



**Figure 6: Help Widows of TUID**

In this window, users can:

- Use the ESC key to go back to the previous window.
- Press the Down or Up key to page down or up.

## 5.6 Save window

When the user presses CTRL-S in the main window, TUID will open the save window.

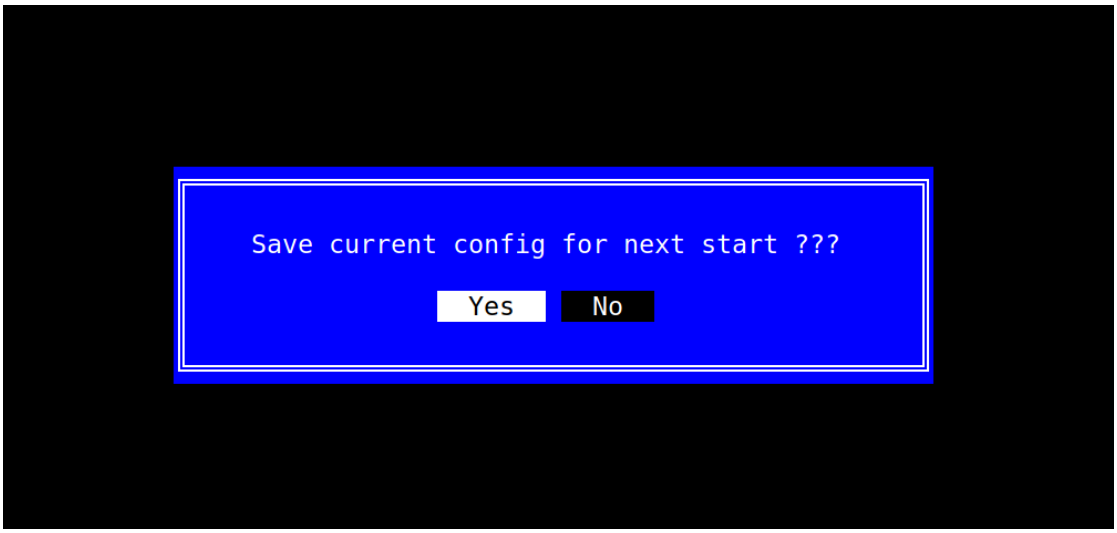



**Figure 7: Save Widow of TUID**

In this window, the user can:

- Use the ESC key to go back to the previous window.
- Press the Left or Right key to switch between Yes or No buttons.
- Press the Enter key to save the configuration or cancel.
- If the current configuration is saved, TUID will load that configuration when it starts next time.

# 6. TROUBLESHOOTING

## 6.1 Terminal Size

TUID will not display the main window normally with the serial console under some Linux systems (e.g. CentOS 7.5). Instead, it will display a blank screen after starting. This issue is because the terminal size of serial console is not set under the system. TUID cannot determine the right size of terminal. The "stty size" command returns "0 0", the COLUMS and LINES are both zero.

To solve this issue, please set the terminal size using the "stty" command before starting TUID and then TUID will display normally.

```
$ stty rows 25
$ stty cols 100
```