|  |  |
| --- | --- |
| **L****ATTICE SEMICONDUCTOR CORPORATION** | THIS DOCUMENT IS THE PROPERTY OF LATTICE SEMICONDUCTOR CORPORATION AND IS ISSUED IN STRICT CONFIDENCE AND SHALL NOT BE REPRODUCED OR DISCLOSED TO THIRD PARTIES. UTILIZATION OF THIS DOCUMENT FOR PURPOSES OTHER THAN THE MANUFACTURE OR USE OF LATTICE PRODUCTS IS PROHIBITED WITHOUT THE EXPRESS CONSENT OF LATTICE SEMICONDUCTOR CORPORATION. |

**AUTHOR & REVIEWERS**

|  |  |  |
| --- | --- | --- |
| TITLE | NAME | DATE |
|  | Jason Wang | 7/7/2021 |
|  | Shawn Yan |  |
|  | Ping Chen |  |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

**DEPARTMENT: EDA CoX**

**DISTRIBUTION**

|  |
| --- |
| All Quality CoX Staffs  LSV:  LSH: Ping Chen |

**REVISION HISTORY**

|  |  |  |
| --- | --- | --- |
| REVISION | RELEASE DATE | COMMENTS |
| 1.0 | 07/18/2021 | Initial Draft |
|  |  |  |
|  |  |  |
|  |  |  |
|  |  |  |

# Table of Contents

[Table of Contents 3](#_Toc77530783)

[1 PURPOSE 7](#_Toc77530784)

[2 INTRODUCTION 7](#_Toc77530785)

[2.1 PROBLEM STATEMENT 7](#_Toc77530786)

[2.2 GOALS 7](#_Toc77530787)

[2.3 TARGET USER 7](#_Toc77530788)

[3 TOP LEVEL INTERFACE 7](#_Toc77530789)

[3.1 Interface Overview 7](#_Toc77530790)

[3.2 Admin Queue Inputs 7](#_Toc77530791)

[3.3 Task Queue Inputs 8](#_Toc77530792)

[3.4 Result Queue outputs 9](#_Toc77530793)

[3.5 Runtime Queue outputs 9](#_Toc77530794)

[3.6 Status Queue outputs 11](#_Toc77530795)

[3.7 Local Suite/Case Inputs 11](#_Toc77530796)

[3.7.1 Local suite file inputs 11](#_Toc77530797)

[3.7.2 Local case inputs 11](#_Toc77530798)

[3.8 Link with other Client 11](#_Toc77530799)

[3.9 Client Console Mode 12](#_Toc77530800)

[3.10 Debug log 13](#_Toc77530801)

[4 GENERAL ARCHITECTURE 13](#_Toc77530802)

[4.1 Management Layer: 14](#_Toc77530803)

[4.2 Implementation Layer: 14](#_Toc77530804)

[4.3 Database Layer: 14](#_Toc77530805)

[5 DETAIL FUNCTION BLOCKS 14](#_Toc77530806)

[5.1 Management Thread 14](#_Toc77530807)

[5.1.1 Initial State 15](#_Toc77530808)

[5.1.2 Work State 15](#_Toc77530809)

[5.1.3 Maintain State 15](#_Toc77530810)

[5.1.4 Stop State 15](#_Toc77530811)

[5.1.5 Initial to Work 16](#_Toc77530812)

[5.1.6 Work to Maintain 16](#_Toc77530813)

[5.1.7 Maintain to Work 16](#_Toc77530814)

[5.1.8 Work to Stop 17](#_Toc77530815)

[5.1.9 Maintain to Stop 17](#_Toc77530816)

[5.1.10 Code Structure 18](#_Toc77530817)

[5.2 View Server 21](#_Toc77530818)

[5.2.1 Inputs 21](#_Toc77530819)

[5.2.2 Outputs 21](#_Toc77530820)

[5.2.3 Tasks 21](#_Toc77530821)

[5.2.4 Code Structure 22](#_Toc77530822)

[5.3 Console Server 22](#_Toc77530823)

[5.3.1 Inputs 23](#_Toc77530824)

[5.3.2 Outputs 23](#_Toc77530825)

[5.3.3 Tasks 23](#_Toc77530826)

[5.3.4 Code Structure 24](#_Toc77530827)

[5.4 Data Server 25](#_Toc77530828)

[5.4.1 Inputs 26](#_Toc77530829)

[5.4.2 Outputs 26](#_Toc77530830)

[5.4.3 Tasks 26](#_Toc77530831)

[5.4.4 Code Structure 26](#_Toc77530832)

[5.5 Link Server 27](#_Toc77530833)

[5.5.1 Inputs 27](#_Toc77530834)

[5.5.2 Outputs 28](#_Toc77530835)

[5.5.3 Tasks 28](#_Toc77530836)

[5.5.4 Code Structure 28](#_Toc77530837)

[5.6 Hall Manager 29](#_Toc77530838)

[5.6.1 Inputs 29](#_Toc77530839)

[5.6.2 Outputs 30](#_Toc77530840)

[5.6.3 Tasks 30](#_Toc77530841)

[5.6.4 Code Structure 30](#_Toc77530842)

[5.7 Tube Server 32](#_Toc77530843)

[5.7.1 Inputs 32](#_Toc77530844)

[5.7.2 Outputs 33](#_Toc77530845)

[5.7.3 Tasks 33](#_Toc77530846)

[5.7.4 Code Structure 34](#_Toc77530847)

[5.8 Misc Timmer 34](#_Toc77530848)

[5.8.1 Inputs 34](#_Toc77530849)

[5.8.2 Outputs 34](#_Toc77530850)

[5.8.3 Tasks 34](#_Toc77530851)

[5.8.4 Code Structure 35](#_Toc77530852)

[6 MAIN WORK FOWS 35](#_Toc77530853)

[6.1 Task Inputting 35](#_Toc77530854)

[6.1.1 Tube server sorting 35](#_Toc77530855)

[6.1.2 Hall Manager sorting 36](#_Toc77530856)

[6.1.3 Task Waiter sorting 36](#_Toc77530857)

[6.2 Environment Build 36](#_Toc77530858)

[6.2.1 Path Build 36](#_Toc77530859)

[6.2.2 Source Export 36](#_Toc77530860)

[6.2.3 Environment setup 37](#_Toc77530861)

[6.3 Result Processing 37](#_Toc77530862)

[7 TESTING & VALIDATION PLAN 38](#_Toc77530863)

[8 REFERENCE DOCUMENTS 38](#_Toc77530864)

[9 DEFINITIONS 38](#_Toc77530865)

[10 GENERAL REQUIREMENTS OR RESPONSIBILITIES 38](#_Toc77530866)

[11 KNOWN LIMITATIONS 38](#_Toc77530867)

[12 ENHANCEMENTS UPDATES 38](#_Toc77530868)

[12.1 <Subject> version <vX.X> 38](#_Toc77530869)

[12.2 <Subject> version <vX.X+1> 38](#_Toc77530870)

[13 APPENDIX 38](#_Toc77530871)

# PURPOSE

This document specifies the overall and detail information for TMP client including the inputs, how to process this information and how to output then.

# INTRODUCTION

## PROBLEM STATEMENT

## GOALS

Introduce TMP client top level interface and its main architecture.

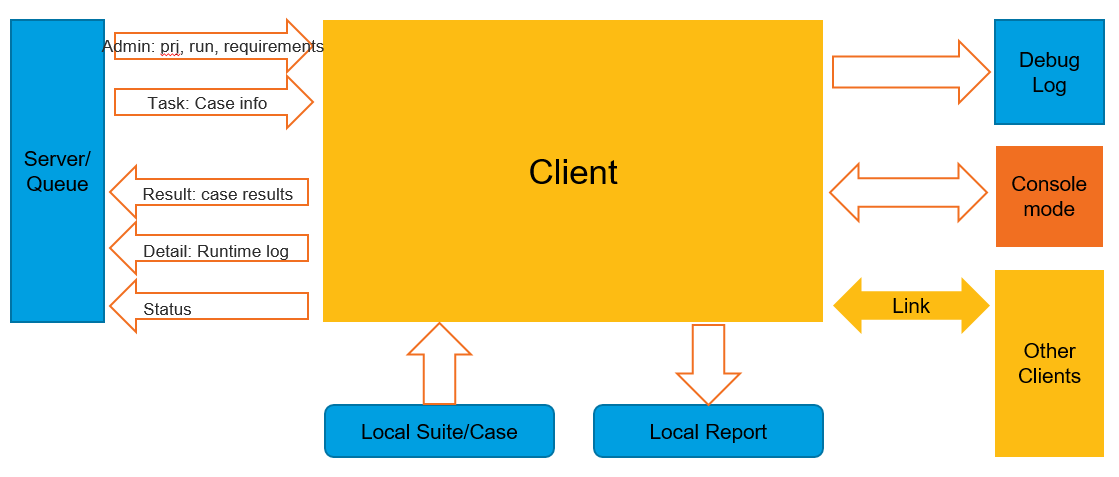
## TARGET USER

New Platform DEV

# TOP LEVEL INTERFACE

## Interface Overview

TMP client will connect with TMP server and local components shown in following pictures:



## Admin Queue Inputs

Admin queue used to pass suite level info to client in xml format. Admin queue contains: project info, run environments and launch command.

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

<!-- sent time: Fri Mar 3 15:48:35 2017 -->

<AdminQ title="run\_778\_021417\_171853">

<ID>

<sub name="project" value="3"></sub>

<sub name="run" value="778"></sub>

<sub name="suite" value="404"></sub>

</ID>

<CaseInfo>

<sub name='suite\_path' value='sim\_suite'></sub>

<sub name='repository' value='http://lshlabd0001/designpool/trunk'></sub>

</CaseInfo>

<Machine>

<sub name='group' value='rna\_regression\_group'></sub>

</Machine>

<Software>

<sub name='diamond' value='3\_10a.13'></sub>

</Software>

<LaunchCommand>

<sub name='cmd' value='python DEV/bin/run\_diamond.py'></sub>

</LaunchCommand>

<System>

<sub name='os\_type' value='windows'></sub>

</System>

<Status>

<sub name="admin\_status" value="processing"></sub>

</Status>

</AdminQ>

## Task Queue Inputs

Task queue used to pass case level info to client in xml format. Task queue contains: case info, run environments and launch command.

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

<Test title="AVID\_C125026\_SwissRoll\_sap">

<TestID>

<sub name="id" value="1314148"></sub>

</TestID>

<CaseInfo>

<sub name='design\_name' value='Conversion\_Branch/sapphire/AE\_suite/AVID\_C125026\_SwissRoll\_sap'></sub>

</CaseInfo>

<LaunchCommand>

<sub name='cmd' value='--synthesis=lse --check-conf=designpool\_lse.conf'></sub>

</LaunchCommand>

</Test>

## Result Queue outputs

Result queue used to pass test case results to TMP server in xml format. Result queue contains: case info, run result and run location.

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

<CaseResults ip="192.168.123.40" machine="LSHITD0097">

<case>

<reason>NA</reason>

<suiteId>2910</suiteId>

<run\_time>0h 2m 11s</run\_time>

<key\_check>NA</key\_check>

<scan\_result></scan\_result>

<defects\_history>NA</defects\_history>

<update\_time>10:19:23 04/13</update\_time>

<milestone>NA</milestone>

<design>jedi\_d1\_M29/Large/01\_adpcm\_codec\_64ch\_70K</design>

<defects></defects>

<testId>18536935</testId>

<location>D:/tmp\_work/results/prj8/run102042/T18536935</location>

<runId>102042</runId>

<projectId>8</projectId>

<status>Processing</status>

</case>

</CaseResults>

## Runtime Queue outputs

Runtime queue used to pass test case runtime console to TMP server in xml format. Runtime queue contains: test case runtime console output logs.

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

<root>

<case>

<suiteId>2683</suiteId>

<runLog>

####################

Run with TMP client:2.12.42

Source Location(Case URL) ==&amp;gt; http://lsh-tmp/platform/trunk/bqs\_scripts/regression\_suite/radiant\_suite/general\_flow/run\_synthesis

Runtime Location(Launch Path) ==&amp;gt; LSHITD0097:D:/tmp\_work/results/prj8/run100152/T18215914

Save location 1 for (Win) access ==&amp;gt; &amp;lt;a href=file:///D:/tmp\_work/results/prj8/run100152/T18215914 target='\_explorer.exe'&amp;gt;D:/tmp\_work\results\prj8\run100152\T18215914&amp;lt;/a&amp;gt;

Note:

1. If the link above not work, please copy it to your file explorer manually.

2. For windows, we can also use \\machine\Disk\_Partition$\run\_path to access directly.

Environments :{EXTERNAL\_RADIANT\_PATH=C:/lscc/radiant/3.0, PYTHONUNBUFFERED=1, EXTERNAL\_PYTHON\_PATH=C:/Python27, EXTERNAL\_DEV\_PATH=G:/repository/tmp\_client/tools/corescripts2/DEV}

LaunchCommand:C:/Python27/python G:/repository/tmp\_client/tools/corescripts2/DEV/bin/run\_radiant.py --synthesis=lse --run-synthesis --design=run\_synthesis/

LaunchDir:D:/tmp\_work/results/prj8/run100152/T18215914

\*---------------------------------------------

\* Welcome to Lattice Batch-queuing System Test Suite

\* HOST NAME: LSHITD0097

\* Platform: Windows-10-10.0.19041

\* Scripts Version: 3235

\* Play Time: 19:05:44 04/12/2021 UTC+8

\*--------------------------------------------

\* SRC DESIGN: D:/tmp\_work/results/prj8/run100152/T18215914/run\_synthesis

\* JOB DESIGN: D:\tmp\_work\results\prj8\run100152\T18215914\run\_synthesis/\_scratch

pnmainc path:

- C:\lscc\radiant\3.0\bin\nt64/pnmainc.exe

Running Command: pnmainc synthesis\_flow.tcl [Mon Apr 12 19:05:44 2021]

&amp;lt;D:\tmp\_work\results\prj8\run100152\T18215914\run\_synthesis\\_scratch&amp;gt;

Status: Pass. Elapsed Time: 5.94 seconds

\*---------------------------------------------

\* Finished Lattice Batch-queuing System Test Flow

\* Stop Time: 19:05:53 04/12/2021 UTC+8

\*

\* Elapsed time: 9 seconds.

\*---------------------------------------------

Launching C:/Python27/python.exe G:/repository/tmp\_client/tools/corescripts2/DEV/tools/check/check.py --report-path=D:/tmp\_work/results/prj8/run100152/T18215914 --vendor radiant --top-dir=D:/tmp\_work/results/prj8/run100152/T18215914 --tag=\_scratch --design=run\_synthesis/ --report=check\_flow.csv --preset-options=check\_normal --rerun-path=D:/tmp\_work/results/prj8/run100152/T18215914

- Final check status: 200

- Passed

</runLog>

<testId>18215914</testId>

<runId>100152</runId>

<projectId>8</projectId>

</case>

</root>

## Status Queue outputs

Status queue used to pass client status to TMP server in xml format. Status queue contains: Host machine CPU, MEM status.

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

<client>

<host\_ip>192.168.123.40</host\_ip>

<os>windows10</os>

<used\_thread>0/3</used\_thread>

<group\_name>rna\_regression\_group</group\_name>

<unattended\_mode>0</unattended\_mode>

<task\_take></task\_take>

<riviera>2018.10</riviera>

<memory\_used>39</memory\_used>

<squish>6.1.0 6.6.2</squish>

<core\_version>3371</core\_version>

<radiant>ng2\_2 ng3\_0 sd\_ng2\_2p.94 sd\_ng222</radiant>

<disk\_left>376</disk\_left>

<cpu\_used>4</cpu\_used>

<diamond>3.11.446 3.12</diamond>

<modelsim>10.1</modelsim>

<os\_type>windows</os\_type>

<private\_mode>1</private\_mode>

<questasim>10.6</questasim>

<activehdl>NA</activehdl>

<client\_version>2.12.52</client\_version>

<icecube>NA</icecube>

<host\_name>LSHITD0097</host\_name>

<account>jwang1</account>

<status>Free</status>

</client>

## Local Suite/Case Inputs

Client also support local test suite file or test case inputs.

### Local suite file inputs

Local suite file is an Excel file which contains: suite sheet and case sheet. Suite sheet used to describe suite level run information and case level used to describe case level information.

You may find a demo in client install path: doc/TMP\_example/standard\_suite/diamond\_regression.xlsx

### Local case inputs

Client can run a standard case directly.

You may find a demo in client install path: doc/TMP\_example/standard\_case/case1

## Link with other Client

Client was designed to link with other client for future distribution task support. This kind of link between clients was construct by socket message.

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

<slave\_data ip="192.168.122.58" machine="D27639">

<task>

<sub name='request' value='FINISHED'></sub>

</task>

<info>

<sub name='request' value='SYSTEM'></sub>

</info>

<action>

<sub name='request' value='shutdown/restart'></sub>

</action>

</slave\_data>

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

<master\_data ip="192.168.122.58" machine="D27639">

<results>

<sub name='default' value='NA'></sub>

</results>

</master\_data>

## Client Console Mode

Client console mode used to developer maintenance, this maintenance object could be local or remote client. In maintenance mode, following background data can be found: host machine information, Client task information. Also some client/host machine restart/shutdown action supported.

Console mode start by: client/clientc.exe -I

For more information please type ‘help’ in console mode.

==========Welcome to TMP Client Interactive Mode==========

->help

TMP Client Interactive Mode Commands:

H -- Show all commands.

HELP -- Show all commands.

I -- Show linked host client info.

INFO -- Show linked host client info.

T -- Show linked Client task info.

TASK -- Show linked Client task info.

A -- Show linked Client available actions.

ACTION -- Show linked Client available actions.

D -- Show linked Client database info.

DATABASE -- Show linked Client database info.

TH -- Show linked Client threads status.

THREAD -- Show linked Client threads status.

L -- Link to remote machine, default is localhost.

LINK -- Link to remote machine, default is localhost.

E -- Exit Client Inteactive Mode.

EXIT -- Exit Client Inteactive Mode.

You may type: '<command> help' for more info

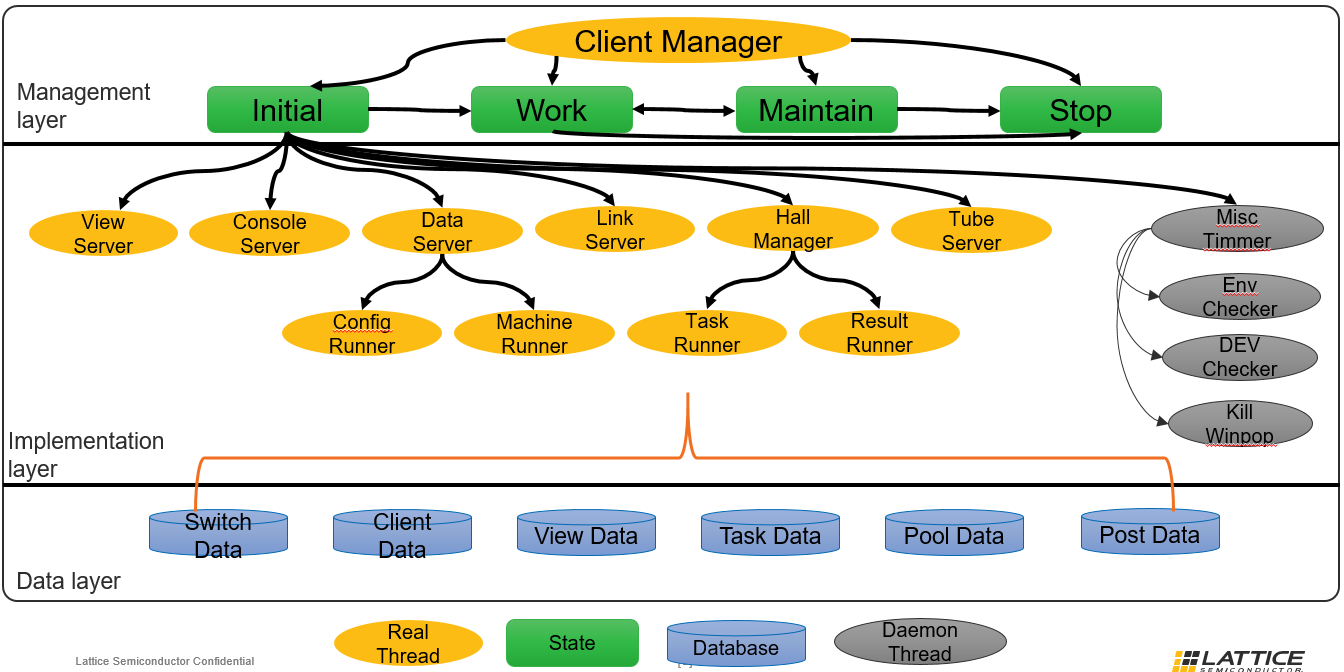
## Debug log

Client can be run in debug mode which will dump following information:

1. Input admin/task queue message
2. Output results/Console log/status message
3. Host machine CPU, MEM, Space info

# GENERAL ARCHITECTURE

Client was designed as three levels to get a well multiple thread performance.



## Management Layer:

Client was designed to be sensitive on host machine status, remote corescript status, remote client update and can be switch between different states. Will details the switch conditions in following sections.

## Implementation Layer:

This layer will be responsible for all client related tasks. Including: GUI data service, Console service, Client data collection, Task run service, tube service…

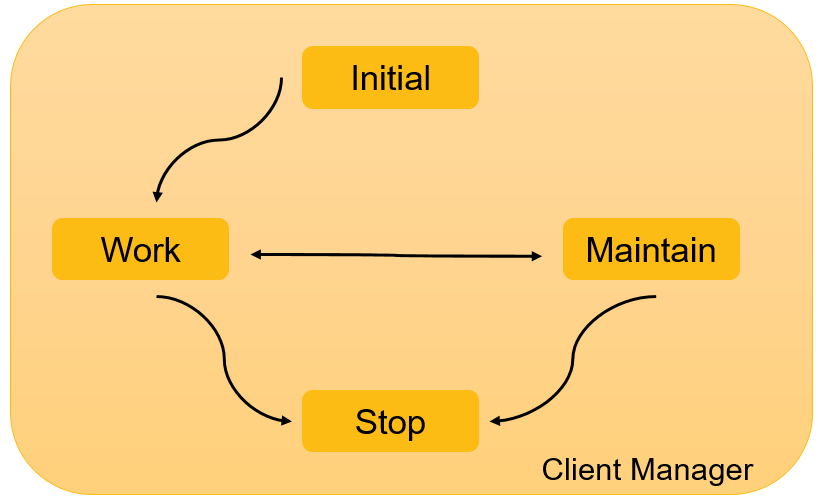
## Database Layer:

Client has lots of data to be read/write concurrently, that’s why we have individual database created for multiple threads access.

# DETAIL FUNCTION BLOCKS

## Management Thread

Client integrated a state machine on design top which include following state inside: initial, work, maintain and stop. ‘client manager’ thread will check the current status of host machine, remote script status, work load and switch between different states.



### Initial State

#### Run Things

* NA

### Work State

#### Run Things

* NA

### Maintain State

#### Run Things

* Corescript update if available
* Client self-update if available
* Environment issue report
* Host machine lower CPU usage waiting
* Host machine lower MEM usage waiting
* Host machine Space overload issue report
* Work space update

### Stop State

#### Run Things

* Stop user interface
* Stop ‘link server’
* Stop daemon process
* Dump processed queue report
* Dump finished queue data
* Send mail if client crashed
* Create new client instance if restart

### Initial to Work

#### Transition condition

NA, this transition happens automatically after Client launched.

#### Transition things

After client launched following things will be launched:

* Start data server for client data collection
* Launch user interface which can be: GUI, Console or Interactive mode.
* Launch daemon process like: Environment check, corescript check, popup window killer (on windows)
* Client update if new version available
* Start link server for client connection support
* Start tube server for RabbitMQ connection support
* Start hall manager for internal task prepare, launch and report

### Work to Maintain

#### Transition condition

Top level ‘Client manager’ will check client current status every 5 seconds. Client will go into ‘Maintain’ state when any of following event happens:

* System idle: No task/case run for more than 5 minutes
* Corescript update available
* Environment issue: Python tool issue, Python environment issue, work space read/write issue
* Host machine CPU overload: host machine CPU usage higher than 95%
* Host machine MEM overload: host machine MEM usage higher than 98%
* Host machine Space overload, depends on user configuration
* Work space change: work space update to another place

#### Transition things

Following things will be run before client go into new state

* Pause ‘hall manager’

### Maintain to Work

#### Transition condition

Top level ‘Client manager’ will make client go into Work state after finish following things:

* Corescript update if available
* Client self-update if available
* Environment issue fixed
* Host machine CPU usage less than 95% or timeout
* Host machine MEM usage less than 98% or timeout
* Host machine Space overload issue fixed
* Work space changed

#### Transition things

Following things will be run before client go into new state

* Start ‘hall manager’

### Work to Stop

#### Transition condition

Top level ‘Client manager’ will check client current status every 5 seconds. Client will go into ‘Stop’ state when any of following event happens:

* User GUI software exit
* Software core dump
* Software self-update exit
* Client restart now exit
* Client restart later exit
* Client shutdown now exit
* Client shutdown later exit
* Host machine restart now exit
* Host machine restart later exit
* Host machine shutdown now exit
* Host machine shutdown later exit

#### Transition things

Following things will be run before client go into new state

* Wait all current run task/case finished
* Stop ‘hall manager’
* Stop ‘tube server’
* Stop ‘data server’

### Maintain to Stop

#### Transition condition

Top level ‘Client manager’ will check client current status every 5 seconds. Client will go into ‘Stop’ state when any of following event happens:

* User GUI software exit
* Software core dump
* Software self-update exit
* Client restart now exit
* Client restart later exit
* Client shutdown now exit
* Client shutdown later exit
* Host machine restart now exit
* Host machine restart later exit
* Host machine shutdown now exit
* Host machine shutdown later exit

#### Transition things

Following things will be run before client go into new state

* Wait all current run task/case finished
* Stop ‘hall manager’
* Stop ‘tube server’
* Stop ‘data server’

### Code Structure

#<file>::<function>

//.java

client\_manager::monitor\_run()

#initial

::to\_work\_status() //by default the initial state is ‘INITIAL’

initial\_status::get\_client\_data\_ready()

initial\_status::launch\_user\_interface()

initial\_status::get\_daemon\_process\_ready()

initial\_status::get\_client\_self\_update()

initial\_status::release\_corescript\_msg()

initial\_status::release\_auto\_restart\_msg()

initial\_status::launch\_link\_services()

initial\_status::local\_console\_run\_recognize()

initial\_status::get\_tube\_server\_ready()

initial\_status::get\_hall\_manager\_ready()

initial\_status::client.set\_current\_status(client.WORK)

#loop

::impl\_current\_info\_update()

::if(start\_work\_mode(client\_sts))::client\_sts.to\_work\_status()

::if(start\_maintenance\_mode(client\_sts))::client\_sts.to\_maintain\_status()

::if(start\_stop\_mode(client\_sts))::client\_sts.to\_stop\_status() ()

::client\_sts.do\_state\_things()

=========================state details=========================

#initial\_status

::to\_work()

::get\_client\_data\_ready()

::launch\_user\_interface()

::get\_daemon\_process\_ready()

::get\_client\_self\_update()

::release\_corescript\_msg()

::release\_auto\_restart\_msg()

::launch\_link\_services()

::local\_console\_run\_recognize()

::get\_tube\_server\_ready()

::get\_hall\_manager\_ready()

::client.set\_current\_status(client.WORK)

::to\_maintain()

::client.set\_current\_status(client.MAINTAIN)

::to\_stop()

::client.set\_current\_status(client.STOP);

::do\_state\_things()

NA

#maintain\_status

::to\_work()

::client.hall\_runner.wake\_request();

::client.set\_current\_status(client.WORK)

::to\_maintain()

NA

::to\_stop()

::client.hall\_runner.soft\_stop()

::client.tube\_runner.soft\_stop()

::client.data\_runner.soft\_stop()

::client.set\_current\_status(client.STOP)

::do\_state\_things()

#maintain\_entry == idle

::implements\_self\_quiet\_update()

::implements\_core\_script\_update()

::implements\_auto\_restart\_action()

#maintain\_entry == update

::implements\_core\_script\_update()

#maintain\_entry == environ

::implements\_env\_issue\_propagate()

::send\_environ\_issue\_mail()

#maintain\_entry == cpu

:: implements\_client\_cpu\_action()

#maintain\_entry == mem

::implements\_client\_mem\_action()

#maintain\_entry == spaces

::implements\_client\_space\_action()

#maintain\_entry == workspace

:: implements\_work\_space\_update()

#work\_status

::to\_work()

NA

::to\_maintain()

:: client.hall\_runner.wait\_request()

::client.set\_current\_status(client.WORK)

::to\_stop()

::client.hall\_runner.soft\_stop()

::client.tube\_runner.soft\_stop()

::client.data\_runner.soft\_stop()

::client.set\_current\_status(client.STOP)

::do\_state\_things()

NA //things already running by other threads

#stop\_status

::to\_work()

NA

::to\_maintain()

NA

::to\_stop()

NA

::do\_state\_things()

::stop\_user\_interface()

::stop\_link\_services()

::stop\_daemon\_process()

export\_data::export\_disk\_processed\_queue\_report()

export\_data::export\_disk\_finished\_queue\_data()

export\_data::export\_disk\_memory\_queue\_data()

::send\_exception\_report()

::restart\_shutdown\_run()

::stop\_with\_exit\_state()

## View Server

View server used to support GUI panel data update, user GUI actions like retest, stop, pause, play and background message prompt

### Inputs

#### User GUI actions

* User test case ‘retest’ from GUI
* User task queue ‘play’ from GUI
* User task queue ‘pause’ from GUI
* User task queue ‘stop’ from GUI
* User task queue ‘delete’ from GUI

#### Background events

* Client space issue sensed
* Client environment issue sensed
* Remote corescript update sensed

### Outputs

#### GUI Dialogs

* Client space issue dialog
* Client environment issue dialog
* Remote corescript update dialog

#### Task data for View

All following data will be put into ‘view\_data’ for future extract.

* Rejected Queue data
* Captured Queue data
* User selected Task Queue data

### Tasks

View server run following tasks during thread started:

* run\_system\_client\_insts\_check()
* start\_progress\_show()
* start\_main\_gui()

View server run following tasks every 5 seconds:

* implements\_retest\_task\_request()
* implements\_run\_action\_request()
* implements\_user\_del\_request()
* implements\_message\_prompt(top\_view)
* implements\_rejected\_queue\_data\_update()
* implements\_captured\_queue\_data\_update()
* implements\_working\_queue\_data\_update()

### Code Structure

#<file>::<function>

//.java

view\_server::monitor\_run()

#initial

::run\_system\_client\_insts\_check()

::start\_progress\_show()

start\_progress::start()

::start\_main\_gui()

main\_frame::gui\_constructor()

main\_frame::set\_size\_location()

main\_frame::set\_default\_font()

main\_frame::set\_system\_look\_feel()

main\_frame::initial\_components()

main\_frame::new menu\_bar()

menu\_bar::construct\_file\_menu()

menu\_bar::construct\_view\_menu()

menu\_bar::construct\_run\_menu()

menu\_bar::construct\_tool\_menu()

menu\_bar::construct\_setting\_menu()

menu\_bar::construct\_control\_menu()

menu\_bar::construct\_help\_menu()

main\_frame::new work\_panel()

work\_panel::new queue\_panel()

main\_frame::new status\_bar()

main\_frame::launch\_system\_tray()

#loop

::implements\_retest\_task\_request()

::implements\_run\_action\_request()

::implements\_user\_del\_request()

::implements\_message\_prompt(top\_view)

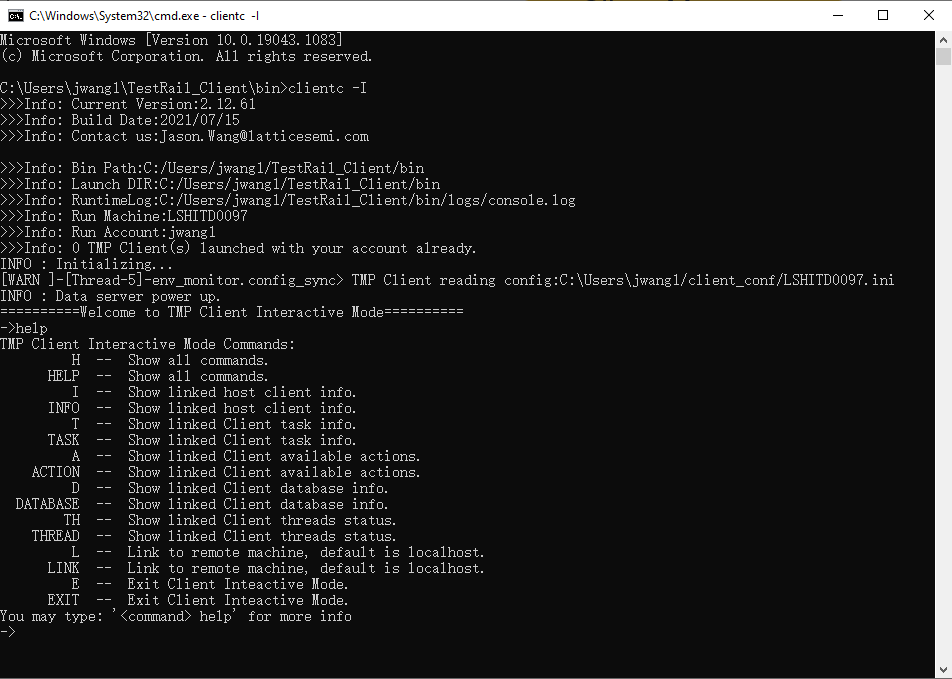
::implements\_rejected\_queue\_data\_update()

::implements\_captured\_queue\_data\_update()

::implements\_working\_queue\_data\_update()

## Console Server

Console server used to create a console for user interface. This user interface allow user check the client data and control the connected client.



### Inputs

User inputs from command line as following:

* ‘Help’ command, to get help information
* ‘info’ command, to get client information about environment, system, machine, software, preference and so on
* ‘task’ command, to get client task information about received, captured, rejected, paused, processing, executing, running…
* ‘action’ command, to restart, shutdown linked client
* ‘database’ command, to get background database info
* ‘thread’ command, to get background thread status info
* ‘link’ command, to link a specified client
* ‘exit’ command, to exit client interactive mode

### Outputs

Return message from linked client as following:

* Help message, returned by ‘help’ command
* Linked client environment, system, machine, software, preference message, returned by ‘info’ command
* Linked client task information returned by ‘task’ command

### Tasks

Console server run following tasks every 100 millisecond:

* Wait and check user inputs
* Check user inputs (check\_user\_inputs(user\_inputs))
* Answer user inputs (answer\_user\_inputs(user\_inputs))

### Code Structure

#<file>::<function>

//.java

console\_server::monitor\_run()

#initial

::TerminalBuilder.builder()

jline-3.12.2-SNAPSHOT.jar::Terminal

#loop

::lineReader.readLine()

::check\_user\_inputs(user\_inputs)

::answer\_user\_inputs(user\_inputs)

case HELP:

::help\_command\_answer();

case INFO:

::info\_command\_answer(input\_list)

::info\_help\_command\_output()

::info\_other\_command\_output()

linked\_client::channel\_cmd\_data\_request()

xml\_parser::create\_common\_xml\_string()//out

xml\_parser::get\_common\_xml\_data(outputs)//in

case TASK:

::task\_command\_answer(input\_list)

::task\_help\_command\_output()

::task\_other\_command\_output()

linked\_client::channel\_cmd\_data\_request()

xml\_parser::create\_common\_xml\_string()//out

xml\_parser::get\_common\_xml\_data(outputs)//in

case ACTION:

::action\_command\_answer(input\_list)

::action\_help\_command\_output()

::action\_other\_command\_output()

linked\_client::channel\_cmd\_data\_request()

xml\_parser::create\_common\_xml\_string()//out

xml\_parser::get\_common\_xml\_data(outputs)//in

case DATABASE:

::database\_command\_answer(input\_list)

::database\_help\_command\_output()

::database\_other\_command\_output()

linked\_client::channel\_cmd\_data\_request()

xml\_parser::create\_common\_xml\_string()//out

xml\_parser::get\_common\_xml\_data(outputs)//in

case THREAD:

::thread\_command\_answer(input\_list)

::thread\_help\_command\_output()

::thread\_other\_command\_output()

linked\_client::channel\_cmd\_data\_request()

xml\_parser::create\_common\_xml\_string()//out

xml\_parser::get\_common\_xml\_data(outputs)//in case LINK:

::link\_command\_answer(input\_list)

::link\_help\_command\_output()

::link\_other\_command\_output()

linked\_client::channel\_cmd\_data\_request()

xml\_parser::create\_common\_xml\_string()//out

xml\_parser::get\_common\_xml\_data(outputs)//in

case EXIT:

::exit\_command\_answer();

## Data Server

Data server used to collect host machine data, including: System, software builds, corescript and client preference data.

### Inputs

NA

### Outputs

All following data will be put into ‘client\_data’ for future extract.

* SW build info, like: diamond, radiant, modelsim
* System info, like: OS, OS type, CPU, MEM
* Corescript info, like: version, update time
* Machine info, like: terminal, ip, group, account, private
* Tools info, like: Perl, Python, SVN
* Preference info, like: thread mode, task mode, max threads, auto rerstart…

### Tasks

Data server run following tasks during thread started:

* Start config file read/write thread -- config\_runner.start();
* Start host machine scan thread -- machine\_runner.start();

Data server run following tasks every 10 seconds:

* dynamic\_merge\_build\_data()
* dynamic\_merge\_system\_data()
* update\_client\_current\_status()
* remove\_invalid\_build\_path()
* update\_max\_sw\_insts\_limitation()
* update\_system\_property\_data()
* debug\_dump\_client\_data()

### Code Structure

#<file>::<function>

//.java

data\_server::monitor\_run()

#initial

config\_sync::config\_runner.start()

machine\_sync::machine\_runner.start()

::initial\_merge\_client\_data(cmd\_info)

#loop

::dynamic\_merge\_build\_data()

::get\_multi\_scan\_dir\_build()

::get\_scan\_cmd\_build()

::dynamic\_merge\_system\_data()

::update\_client\_current\_status()

::remove\_invalid\_build\_path()

::update\_max\_sw\_insts\_limitation()

::update\_system\_property\_data()

::debug\_dump\_client\_data()

file\_action::append\_file\_with\_title()

=========================Sub Process Details=========================

##config\_sync

#initial

::read\_ini\_data()

::import\_data\_verification()

#loop

::get\_write\_config\_file()

::dump\_client\_data()

##machine\_sync

#initial

::update\_static\_data()

::import\_data\_verification()

#loop

:: update\_dynamic\_data()

::get\_avail\_space()

::get\_cpu\_usage()

oshi-core-5.6.0.jar::new SystemInfo()

::get\_mem\_usage()

oshi-core-5.6.0.jar::new SystemInfo()

## Link Server

Link server used to provide link service for remote client (another Client) data request

### Inputs

Data request from remote client, this request including: Task, info, link, action

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

<channel\_cmd ip="192.168.122.58" machine="D27639">

<task>

<sub name='request' value='FINISHED'></sub>

</task>

<info>

<sub name='request' value='SYSTEM'></sub>

</info>

<action>

<sub name='request' value='shutdown/restart'></sub>

</action>

</channel\_cmd>

### Outputs

Request data collected by local client. This data will be fill in the xml message like following.

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

<channel\_cmd ip="192.168.122.58" machine="D27639">

<results>

<sub name='default' value='NA'></sub>

</results>

</channel\_cmd>

### Tasks

Link server run following tasks every 100 millisecond:

* process\_socket\_data()

### Code Structure

#<file>::<function>

//.java

link\_server::monitor\_run()

#initial

NA

#loop

::process\_socket\_data()

#data in

new\_data = socket\_in.readLine()

xml\_parser::get\_common\_xml\_data(new\_data.toString())

#data out

##channel\_cmd

::channel\_cmd\_task\_return\_data()

xml\_parser::create\_common\_xml\_string()

::channel\_cmd\_info\_return\_data()

xml\_parser::create\_common\_xml\_string()

::channel\_cmd\_action\_return\_data()

xml\_parser::create\_common\_xml\_string()

::channel\_cmd\_link\_return\_data()

xml\_parser::create\_common\_xml\_string()

::channel\_cmd\_database\_return\_data()

xml\_parser::create\_common\_xml\_string()

::channel\_cmd\_thread\_return\_data()

xml\_parser::create\_common\_xml\_string()

::channel\_cmd\_default\_return\_data()

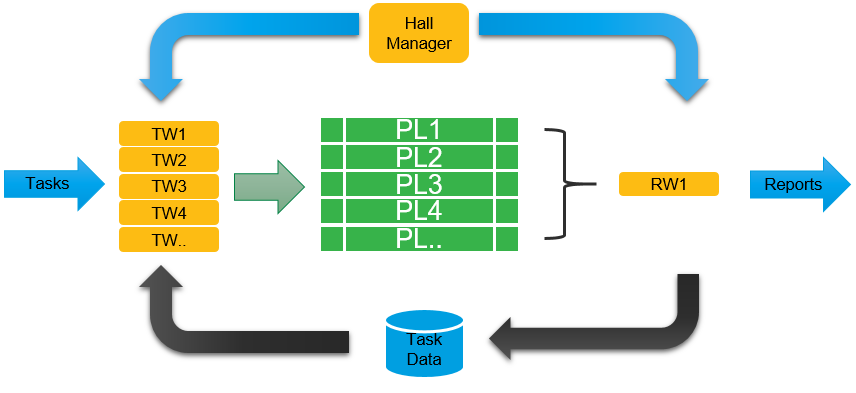
xml\_parser::create\_common\_xml\_string()

##channel\_job

::channel\_job\_default\_return\_data()

## Hall Manager

Hall manager is the core function blocker of Client, this block offer run environment control, report generation and detail task run with integrated threads. In summary, hall manager will start ‘task waiter’ and ‘result waiter’ dynamically based on current work load. ‘task waiter’ threads will take task case from ‘task data’ and setup the run environment then put the run case into thread pool. Task case will be run in the thread pool automatically. ‘result waiter’ will check task case run status in thread pool and report it when task finished.



### Inputs

Admin/Task queues from ‘tube server’:

* task inputs from ‘task data’ processing queue

### Outputs

#### Results outputs

* update\_processed\_task\_data(case\_report\_data);
* send\_tube\_task\_data\_report(case\_report\_data, true);
* send\_tube\_task\_runtime\_report(case\_runtime\_log\_data);

#### Report outputs

* update\_client\_run\_case\_summary(case\_report\_data);
* generate\_console\_report(waiter\_name, case\_report\_data);
* run\_local\_disk\_report(case\_report\_data);

### Tasks

Hall manager run following tasks during thread started:

* get\_task\_waiter\_ready(); -- for task queue processing
* get\_result\_waiter\_ready(); -- for result reporting

Hall manager run following tasks every 10 seconds:

* job\_environment\_build();
* job\_implementation\_monitor();
  + implement\_thread\_auto\_adjustment()-- thread number auto update based on current environment
  + implement\_task\_blocker\_actions()-- pause task queue with lots of case failed
* job\_report\_generation();

### Code Structure

#<file>::<function>

//.java

hall\_manager::monitor\_run()

#initial

::initial\_thread\_pool\_setting()

::get\_task\_waiter\_ready()

::get\_result\_waiter\_ready()

#loop

::job\_environment\_build()

::generate\_executing\_admin\_queue\_list()

::update\_run\_environment()

::job\_implementation\_monitor()

::implement\_thread\_auto\_adjustment()

::implement\_task\_blocker\_actions()

:: job\_report\_generation()

::implement\_general\_console\_report()

::implement\_local\_cmd\_mode\_exit()

=========================Sub Process Details=========================

task\_waiter::monitor\_run()

#initial

::update\_finished\_admin\_queue\_list()

import\_data::import\_disk\_finished\_admin\_queue\_list()

::retrieve\_queues\_into\_memory()

import\_data::retrieve\_disk\_dumped\_received\_admin\_data()

xml\_parser::get\_xml\_file\_admin\_queue\_data()

import\_data::retrieve\_disk\_dumped\_received\_task\_data()

xml\_parser::get\_xml\_file\_task\_queue\_data()

import\_data::retrieve\_disk\_dumped\_processed\_admin\_data()

xml\_parser::get\_xml\_file\_admin\_queue\_data()

import\_data::retrieve\_disk\_dumped\_processed\_task\_data()

xml\_parser::get\_xml\_file\_task\_queue\_data()

#loop

::reload\_repressing\_queue\_data()

::start\_new\_task\_check()

::get\_right\_task\_queue()

task\_data::get\_executing\_admin\_queue\_list()

::get\_runable\_queue\_list()

task\_data::get\_data\_from\_captured\_admin\_queues\_treemap()

client\_data::booking\_used\_soft\_insts()

pool\_data::booking\_reserved\_threads(1)

::get\_final\_task\_data()

::get\_standard\_case\_data()

::get\_merged\_remote\_task\_info()

::merge\_default\_and\_preference\_data()

task\_data::register\_case\_to\_processed\_task\_queues\_map()

task\_prepare::get\_task\_case\_ready()

task\_prepare::get\_task\_path\_ready()

task\_prepare::get\_case\_path\_ready ()

task\_prepare::get\_url\_type()

task\_prepare::get\_zip\_type()

task\_prepare::run\_src\_export()

task\_prepare::run\_src\_unzip()

task\_prepare::get\_script\_path\_ready()

task\_prepare::get\_launch\_command()

task\_prepare::get\_launch\_environment()

::run\_pre\_launch\_reporting()

task\_prepare::dump\_disk\_task\_report\_data()//local report

task\_prepare::send\_tube\_task\_data\_report()//remote report

xml\_parser::create\_result\_document\_string()

rmq\_tube::basic\_send()

rabbitmq-client.jar::basicPublish()

pool\_data::add\_sys\_call()

=========================Sub Process Details=========================

result\_waiter::monitor\_run()

#initial

NA

#loop

::update\_running\_queue\_list()

::update\_finished\_queue\_list()

::dump\_finished\_queue\_report()

::dump\_finished\_queue\_data()

::fresh\_thread\_pool\_data()

pool\_data::fresh\_sys\_call()

post\_data::fresh\_postrun\_call()

::clean\_postrun\_map\_data()

::clean\_history\_send\_data()

::terminate\_user\_request\_running\_call()

::terminate\_stopped\_queue\_running\_task()

::terminate\_local\_user\_request\_running\_task()

::terminate\_remote\_user\_request\_running\_task()

::generate\_case\_report\_data()

task\_report::send\_tube\_task\_data\_report()

task\_report::generate\_optimized\_remote\_send\_data()

xml\_parser::create\_result\_document\_string()

rmq\_tube::basic\_send()

rabbitmq-client.jar::basicPublish()

::generate\_case\_runtime\_log\_data()

task\_report::send\_tube\_task\_runtime\_report()

xml\_parser::create\_result\_document\_string()

rmq\_tube::basic\_send()

rabbitmq-client.jar::basicPublish()

::generate\_console\_report()

::update\_client\_run\_case\_summary()

::update\_processed\_task\_data()

::run\_local\_disk\_report()

Task\_eport::dump\_disk\_task\_report\_data()

::run\_post\_process()

post\_data::add\_postrun\_call()

postrun\_call::run\_process\_cleanup()

<install\_path>/tools/kill\_process.py

postrun\_call::run\_local\_scan\_report\_generate()

postrun\_call::run\_remote\_scan\_report\_generate()

postrun\_call::run\_disk\_cleanup()

task\_report::dump\_disk\_task\_report\_data()

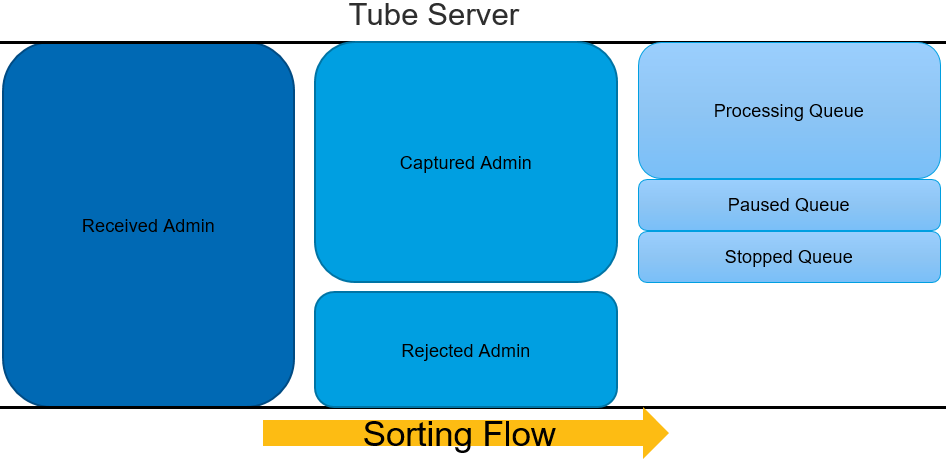
## Tube Server

Tube server is a virtual tunnel used to input/output data.

### Inputs

#### Task queue inputs

Admin queue sorting to catch effective queue for following flow use.



#### Local suite inputs

Local standard suite file (Excel file)

#### Local path inputs

Local standard case suite path

### Outputs

#### Remote Admin Queue

Sorted admin queue data will put into ‘task data’ including following data:

* Processing queue
* Paused queue
* Stopped queue

#### Local Admin Queue

Local suite file/path will be processed and generated local Admin and Task Queue and put into ‘task data’.

### Tasks

Tube server run following tasks during thread started:

* send\_client\_info("complex”) -- send client initial status info
* update\_cmd\_suites\_to\_task\_data(cmd\_info); -- integrated test suite/path from command line

View server run following tasks every 5 seconds:

* run\_remote\_tubes\_control()
* run\_import\_local\_file\_admin()
* run\_import\_local\_path\_admin()
* run\_received\_admin\_sorting()
* flash\_tube\_output()
* update\_captured\_queue\_detail\_lists()
* send\_client\_current\_info()

### Code Structure

#<file>::<function>

//.java

tube\_server::monitor\_run()

#initial

::send\_client\_info()

xml\_parser::create\_client\_document\_string()

rmq\_tube::basic\_send()

rabbitmq-client.jar::basicPublish()

export\_data::debug\_disk\_client\_out\_status()

update\_cmd\_suites\_to\_task\_data()

task\_data::update\_local\_file\_imported\_task\_map()

task\_data::update\_local\_path\_imported\_task\_map()

#loop

::run\_remote\_tubes\_control()

::run\_import\_local\_file\_admin()

local\_tube::generate\_suite\_file\_local\_admin\_task\_queues()

::run\_import\_local\_path\_admin()

local\_tube::generate\_suite\_path\_local\_admin\_task\_queues()

::run\_received\_admin\_sorting()

::flash\_tube\_output()

task\_data::set\_captured\_admin\_queues\_treemap()

task\_data::set\_rejected\_admin\_reason\_treemap()

::update\_captured\_queue\_detail\_lists()

task\_data::set\_processing\_admin\_queue\_list()

task\_data::set\_paused\_admin\_queue\_list()

task\_data::set\_stopped\_admin\_queue\_list()

::send\_client\_current\_info()

xml\_parser::create\_client\_document\_string()

rmq\_tube::basic\_send()

rabbitmq-client.jar::basicPublish()

## Misc Timmer

‘Misc timmer’ involved following threads and start then every predicted time.

* ‘Env Checker’ -- environment check thread
* ‘DEV Checker’ – remote corescript check thread
* ‘Kill winpop’ – windows popup window check thread

### Inputs

NA

### Outputs

NA

### Tasks

Run following tasks:

Start ‘Env Checker’ after 0 second, and run it every 10 seconds.

Start ‘DEV Checker’ after 3 seconds, and run it every 10 seconds.

Start ‘Kill winpop’ after 6 seconds, and run it every 10 seconds.

### Code Structure

#<file>::<function>

//.java

misc\_timer.scheduleAtFixedRate(new env\_checker())

env\_checker::python\_version\_monitor()

env\_checker::run\_environ\_monitor()

env\_checker::remote\_corescript\_monitor()

misc\_timer.scheduleAtFixedRate(new dev\_checker())

dev\_checker::update\_remote\_corescript\_info()

dev\_checker::generate\_update\_request()

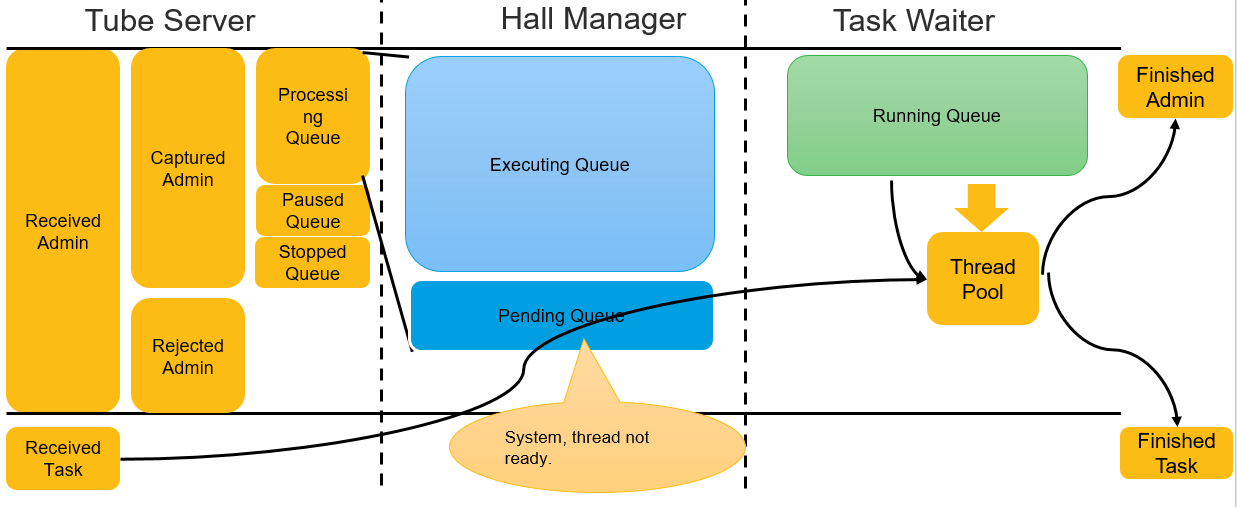
misc\_timer.scheduleAtFixedRate(new kill\_winpop()) //only for windows

<install\_path>/tools/kill\_winpop.py

# MAIN WORK FOWS

## Task Inputting

Admin queue from TMP server have some specific requirements which may not suitable for current client. So there is a Admin queue sorting flow to find suitable queue for client use.



### Tube server sorting

‘tube server’ will catch all admin queues form TMP server and put all received queue into ‘Received Admin’. Based on client available value ‘tube server’ will sorting received admin queue into ‘Captured Admin’ and ‘Rejected Admin’ queue data. Finally, ‘tube server’ will generate following queue list for future use:

### Hall Manager sorting

Hall manager extract all ‘processing queue’ from ‘task data’ and check current system and thread status, if current environment is suitable for a processing queue, this queue will be put into ‘executing queue’.

### Task Waiter sorting

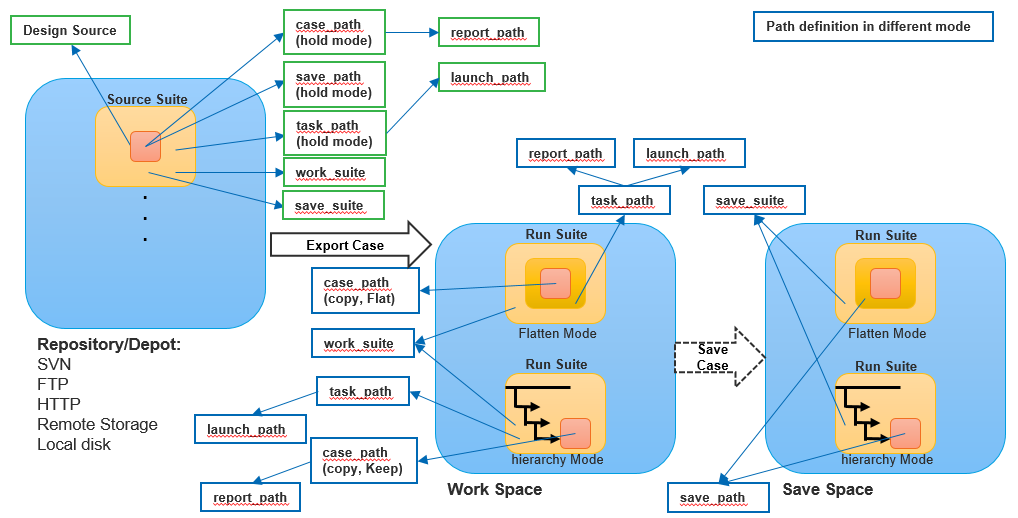
Task Waiter will extract runnable admin queue based on current system resource (available Software, thread) and put this queue into running queue. Also Based on current running queue (‘Admin Queue’), ‘task waiter’ will extract task case from accordingly ‘Task Queue’, prepare the run environment and put this case into ‘Thread Pool’.

## Environment Build

Before Task case could be run in Thread Pool, some extra things need to be run: run path build, case export from SVN, script export (if need) and launch environment, command line build.

### Path Build

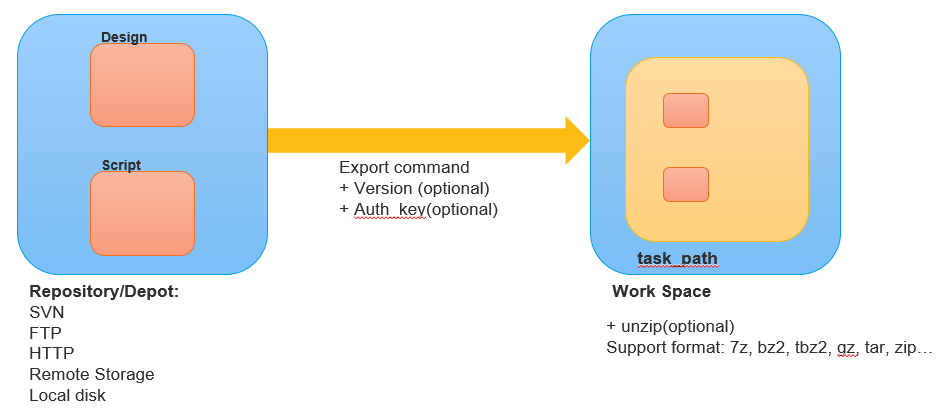
Client has multiple run mode and task case could be in multiple place, so ‘launch path’, ‘report path’ and ‘save path’ could be different from each other. Client will build these path based on current run mode. For details please check following picture:



### Source Export

Client will get source info from Task queue which could have following elements:

|  |  |
| --- | --- |
| CaseInfo | repository |
| suite\_path |
| design\_name |
| version |
| durl\_type |
| dzip\_type |
| script\_url |
| script\_version |
| surl\_type |
| szip\_type |
| auth\_key |
|
|

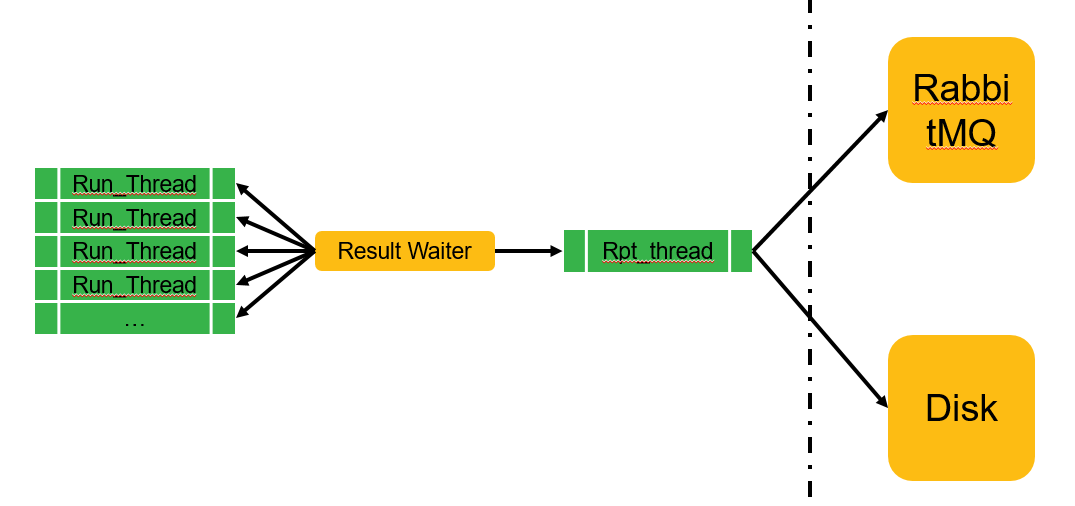


### Environment setup

Before client launch task case, Environment pass down from Task Queue will be setup.

## Result Processing

Result processing constructed by: job thread pool, ‘result\_waiter’ and post job thread pool.



# TESTING & VALIDATION PLAN

# REFERENCE DOCUMENTS

# DEFINITIONS

# GENERAL REQUIREMENTS OR RESPONSIBILITIES

# KNOWN LIMITATIONS

# ENHANCEMENTS UPDATES

## <Subject> version <vX.X>

This outlines all changes and/or new enhancements for this particular version.

## <Subject> version <vX.X+1>

This outlines all changes and/or new enhancements for this particular version.

# APPENDIX