Quali



TestCenter Controller Shell

Release date: January 2019

Shell version 1.5.3

Document version A

***Legal notice***

*Information in this document is subject to change without notice. Without limiting the rights under copyright, no part of this document may be reproduced, stored in or introduced into a retrieval system, or transmitted in any form or by any means (electronic, mechanical, photocopying, recording, or otherwise), or for any purpose, without the express written permission of Quali Ltd. Quali may have patents, patent applications, trademarks, copyrights, or other intellectual property rights covering subject matter in this document. Except if expressly provided in any written license agreement from Quali, the furnishing of this document does not give you any license to these patents, trademarks, copyrights, or other intellectual property. Quali, CloudShell, CloudShell Authoring, CloudShell Resource Manager, CloudShell Remote Runner, CloudShell Runtime, CloudShell Monitor, CloudShell Spy, CloudShell Portal, the Quali logo, the CloudShell logo, and the CloudShell application logos, and all other Quali product names and logos are trademarks or registered trademarks of Quali Ltd. The absence of a trademark from this list does not constitute a waiver of Quali intellectual property rights concerning that trademark.*

*All other trademarks, brand and product names are property of their respective holders.*

*© 2016 Quali Ltd. All rights reserved.*

Contents

[Overview 3](#_Toc518503130)

[About TestCenter Controller Shell 3](#_Toc518503131)

[Standard version 3](#_Toc518503132)

[Supported OS 3](#_Toc518503133)

[Requirements 3](#_Toc518503134)

[Downloading the Shell 3](#_Toc518503135)

[Import and Configure the Shell 4](#_Toc518503136)

[Importing the Shell into CloudShell 4](#_Toc518503137)

[Offline installation of a Shell 4](#_Toc518503138)

[Configuring a new service 5](#_Toc518503139)

[Updating Python Dependencies for Shells 7](#_Toc518503140)

[Updating offline Python dependencies 7](#_Toc518503141)

[Updating online Python dependencies 7](#_Toc518503142)

[Typical workflow 8](#_Toc518503143)

[Scenario 1 – Use a controller to run TestCenter traffic 8](#_Toc518503144)

[Release notes 15](#_Toc518503145)

# Overview

A Shell implements integration of a device model, application or other technology with CloudShell. A shell consists of a data model that defines how the device and its properties are modeled in CloudShell, along with automation that enables interaction with the device via CloudShell.

## About TestCenter Controller Shell

This Shell provides you with connectivity and management capabilities such as loading configuration, running traffic and getting results for TestCenter application.

## Standard version

The TestCenter Controller Shell 1.3.0 is based on the Traffic Shell standard version 3.0.0.

For detailed information about the Shell’s structure and attributes, see the Traffic Shell standard on [cloudshell-standards repository](https://github.com/QualiSystems/shell-traffic-standard) in GitHub.

## Supported OS

* Windows

## Requirements

* CloudShell version 7.1 and above
* TestCenter rest server – either lab server or stc webservices.

## Downloading the Shell

The TestCenter Controller Shell is available from the [Quali Developer Center](http://community.quali.com/spaces/12/index.html?__hstc=46213176.aaafbe5adb338215377a985e0c025079.1467146361756.1471392182746.1471395614692.11&__hssc=46213176.1.1471395614692&__hsfp=2437115919)[.](https://support.qualisystems.com/entries/87063688-Solution-Pack-Download-Center) Download the files into a temporary location on your local machine.

The Shell comprises:

|  |  |
| --- | --- |
| Spirent\_TestCenter\_controller.zip | The Shell Package. |
| Spirent\_TestCenter\_controller\_offline\_requirements.zip | Shell Python dependencies (**for offline installation only**) |
| TestCenter Controller Shell Doc.pdf | Documentation |

# Import and Configure the Shell

This section describes how to import, configure and modify the TestCenter Controller Shell.

## Importing the Shell into CloudShell

Use the following procedure to import the downloaded Shell:

**To import the Shell into CloudShell:**

1. Download the Shell from the [Quali Developer Center](http://community.quali.com/spaces/12/index.html?__hstc=46213176.aaafbe5adb338215377a985e0c025079.1467146361756.1471392182746.1471395614692.11&__hssc=46213176.1.1471395614692&__hsfp=2437115919)[.](https://support.qualisystems.com/entries/87063688-Solution-Pack-Download-Center)
2. Back up your database.
3. Log in to **CloudShell Portal** as administrator and access the relevant domain.
4. From the **User** menu, select **Import Package.**



5. Browse to the location of the downloaded Shell file, select the relevant .zip file and click **Open**. Alternatively, drag the shell’s .zip file into CloudShell Portal.

## Offline installation of a Shell

**Note:** Offline installation instructions apply only if Cloudshell Execution Server has no access to PyPi. You can skip this section if your execution server has access to PyPi. *For additional information, see the online help topic on offline dependencies.*

The Shell uses a variety of Python packages.

**To work in offline mode:**

1. Download the Spirent\_chassis\_shell\_offline\_requirments.zip file (see *Downloading the Shell*).
2. Unzip it to a local repository. Make sure the Execution Server has access to this folder.
3. On the Execution Server machine, in the customer.configfile, add the following key:

|  |  |
| --- | --- |
| <add key="PythonOfflineRepositoryPath" value="repository | |
| full path"/> |  |

Make sure to update the value with the path to the repository containing the unzipped file.

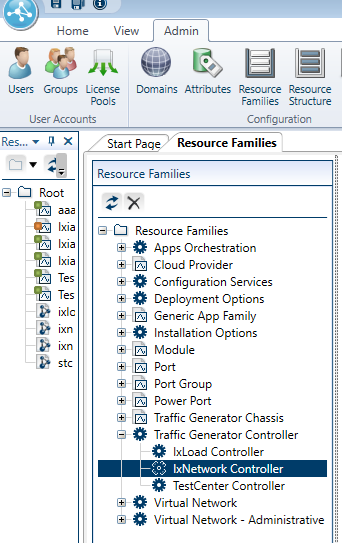
1. Restart the Execution Server.

## Configuring a new service

Perform this procedure to configure the service after importing the Shell.

**To configure the service:**

1. Go to the **Admin** tab and select the **Resource Families**.
2. Under **Traffic Generators Controllers**, select **TestCenter Controller**.



1. Define default values for the **TestCenter Controller** service.

|  |  |
| --- | --- |
| Name | Description |
| Controller Address | The IP address of STC REST server – either lab server or machine running stcweb. |
| Controller TCP Port | The TCP port of STC REST server – either lab server or machine running stcweb. |

# Updating Python Dependencies for Shells

This section guides through on how to update your Python dependencies folder. This is required when you upgrade a Shell, driver that has new/updated dependencies. It applies to both online and offline dependencies.

## Updating offline Python dependencies

**To update offline Python dependencies:**

1. Download the latest Python dependencies package zip file locally.
2. Extract the zip file to the suitable offline package folder(s).
3. Restart any Execution Server that has a live instance of the relevant driver or script. This requires running the TestShell Execution Server's configuration wizard, as explained in the Configure the TestShell Execution Server topic of the CloudShell Suite Installation Guide - see the [CloudShell Docs & Training](http://www.quali.com/community/training/) page.

## Updating online Python dependencies

In online mode, the Execution Server automatically downloads and extracts the appropriate dependencies file to the online Python dependencies repository every time a new instance of the driver or script is created.

**To update online Python dependencies:**

* If there is a live instance of the Shell's driver or script, restart the execution server, as explained above. If an instance does not exist, the execution server will download the Python dependencies the next time a command of the driver or script runs.

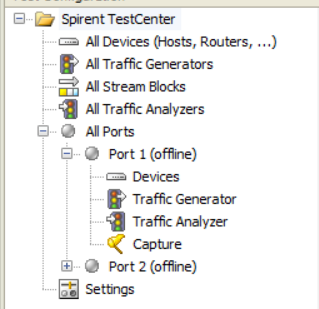
# Typical workflow

## Scenario 1 – Use a controller to run TestCenter traffic

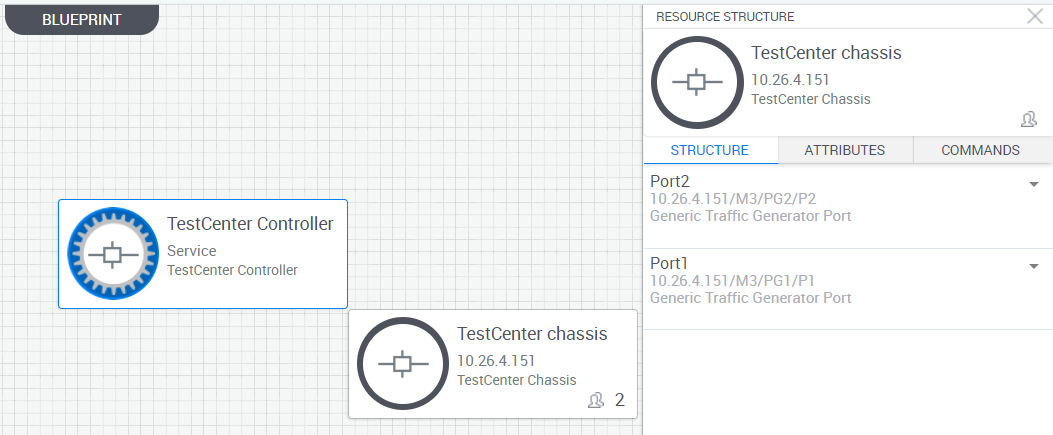
* Create Blueprint

Create a Blueprint with TestCenter controller service and TestCenter chassis resource ports. Number of STC ports in the blueprint should match the number of ports in the TestCenter configuration.

For example, if we have configuration with two ports:



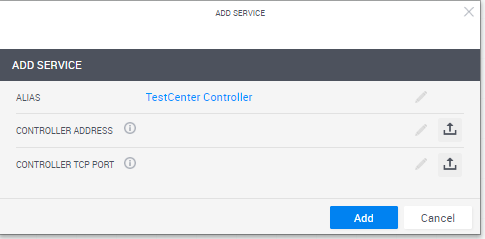
We create Blueprint with two TestCenter ports.



* Reserve Sandbox

Create a Sandbox from the Blueprint.

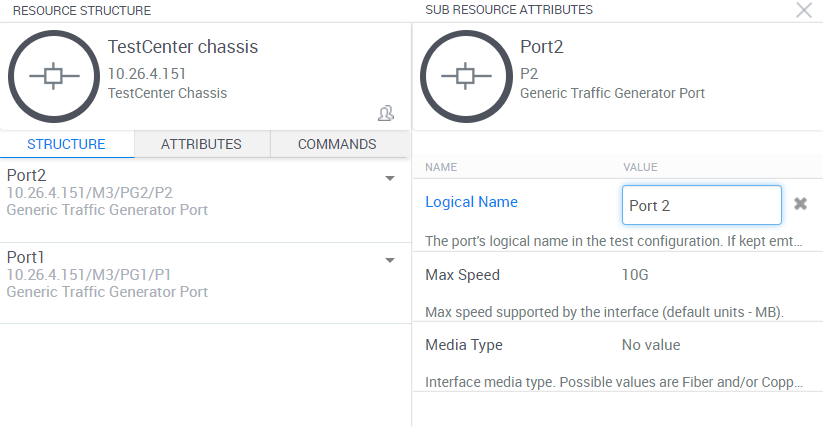
Edit TestCenter Controller Service parameters if required.



See ‘Configuring a new service’ above.

* Map configuration ports to Sandbox ports

For each port in the TestCenter configuration assign physical port from the ports in the sandbox. Open the attributes tab and set the Logical Name to the port name in the configuration.



* Controller Commands

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Description | Parameters | |
|  |  | Parameter | Description |
| Load Configuration | Load configuration and reserve ports | Spirent config file name | Full path to Spirent configuration file name (tcc or xml) |
| Start ARP/ND | Send ARP/ND for all devices and streams |  |  |
| Start All Devices | Start all emulations on all devices |  |  |
| Stop All Protocols | Stop all emulations on all devices |  |  |
| Start Traffic | Start traffic on all ports | Blocking | True - return after traffic finish to run, False - return immediately |
| Stop Traffic | Stop traffic on all ports |  |  |
| Get Statistics | Get view statistics | View Name | generatorPortResults, analyzerPortResults etc. |
| Output Type | CSV or JSON. If CSV. The statistics will be attached to the reservation csv file. |
| Sequencer Command | Perform sequencer command | Command | Start – start sequencer  Stop – stop sequencer  Wait – wait for sequencer to finish |

* Hidden developer commands

|  |  |  |  |
| --- | --- | --- | --- |
| Command | Description | Parameters | |
|  |  | Parameter | Description |
| get\_session\_id | Returns the REST session. This ID can be used to run any STC REST command directly. |  |  |
| get\_children | Returns list of all children of a specific type of the requested object.  If child\_type == None all children will be returned. | obj\_ref | Requested object reference. |
| child\_type | Requested child type |
| get\_attributes | Returns dictionary of all <attribute: value> of the requested object attributes. | obj\_ref | Requested object  reference. |
| set\_attribute | Sets value of specific attribute of the requested object. | obj\_ref | Requested object reference. |
| attr\_name | Requested attribute name. |
| attr\_value | Value to set. |
| perform\_command | Performs any STC command. | command | Requested command (without the Command suffix). |
| parameters\_json | Command parameters dict {name: value} as serialized json. |

* Following is a simple code snippet demonstrating the hidden commands:

The code bellow assumes that reservation\_id holds the reservation ID and session holds CS session.

# Get session ID

session\_id = session.ExecuteCommand(reservation\_id, *'TestCenter Controller'*, *'Service'*, *'get\_session\_id'*)

# Get project object reference.

project = session.ExecuteCommand(reservation\_id, *'TestCenter Controller'*, *'Service'*, *'get\_children'*,

[InputNameValue(*'obj\_ref'*, *'system1'*),

InputNameValue(*'child\_type'*, *'project'*)])

project\_obj = json.loads(project.Output)[0]

# Get all children of project.

project\_childs = session.ExecuteCommand(reservation\_id, *'TestCenter Controller'*, *'Service'*, *'get\_children'*,

[InputNameValue(*'obj\_ref'*, project\_obj)])

# Get automation-options object reference.

options = session.ExecuteCommand(reservation\_id, *'TestCenter Controller'*,

*'Service'*, *'get\_children'*, [InputNameValue(*'obj\_ref'*, *'system1'*),

InputNameValue(*'child\_type'*, *'AutomationOptions'*)])

options\_ref = json.loads(options.Output)[0]

# Get automation-options attributes.

options\_attrs = session.ExecuteCommand(reservation\_id, *'TestCenter Controller'*,

*'Service'*, *'get\_attributes'*, [InputNameValue(*'obj\_ref'*, options\_ref)])

# Set automation-options log-level attribute.

session.ExecuteCommand(reservation\_id,

*'TestCenter Controller'*,

*'Service'*, *'set\_attribute'*, [InputNameValue(*'obj\_ref'*, options\_ref),

InputNameValue(*'attr\_name'*, *'LogLevel'*),

InputNameValue(*'attr\_value'*, *'INFO'*)])

# Perform subscribe command.

parameters = {*'Parent'*: project\_obj,

*'ResultParent'*: project\_obj,

*'ConfigType'*: *'Generator'*,

*'ResultType'*: *'GeneratorPortResults'*}

session.ExecuteCommand(reservation\_id,

*'TestCenter Controller'*,

*'Service'*, *'perform\_command'*, [InputNameValue(*'command'*, *'ResultsSubscribe'*), InputNameValue(*'parameters\_json'*,json.dumps(parameters))])

References

Additional technical documentation is available in the [Quali's Developer Center](http://community.quali.com/spaces/12/index.html?__hstc=46213176.aaafbe5adb338215377a985e0c025079.1467146361756.1471392182746.1471395614692.11&__hssc=46213176.1.1471395614692&__hsfp=2437115919)[.](https://support.qualisystems.com/entries/22858046-download-center)

For Quali discussion forums, click [here](http://community.quali.com/spaces/13/index.html?__hstc=46213176.aaafbe5adb338215377a985e0c025079.1467146361756.1471392182746.1471395614692.11&__hssc=46213176.1.1471395614692&__hsfp=2437115919)[.](https://support.qualisystems.com/)

# Release notes

**What’s new:**

* Do not fail on empty results, return empty table.

**Known issue:**

* NA