

# Running TPT in Docker

Version 18u3



Due to continuous product development, information in this document is subject to change without notice.

No part of this user manual may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording, or by any information storage and retrieval system without express written permission from Piketec GmbH.

TPT Time Partition Testing and TPT logo are registered trademarks of Piketec GmbH.



[www.piketec.com](http://www.piketec.com)

## Table of Contents



<b>1 About this document</b> .....	<b>4</b>
<b>2 Introduction</b> .....	<b>5</b>
<b>3 TPT Docker image</b> .....	<b>9</b>
<b>4 Executable of SUT</b> .....	<b>10</b>
<b>5 TPT Docker container</b> .....	<b>11</b>
5.1 Running tests in a Docker container via command line .....	12
5.2 Running TPT in a Docker container via API server .....	13

# 1 About this document

To facilitate the reading, the text is formatted as follows:

Formatted text	Explanation
FUSION platform, Help view, Declaration Editor, Build Progress dialog	places where you can find something in TPT; names of platforms, views, editors, dialogs, assesslets, step types, dashboard widgets, and tabs
View   Show View   Content	menu path to an item in TPT or to a topic in a PDF manual
<TPT.INSTALLDIR>/ templates	path to files or folders on your computer
Restrict to linked test cases	names of text fields, columns, menu items, buttons, or checkboxes in TPT
Stub header files	link to another topic, bookmark, or to a website
TPT.assertAlways	source code
TestCase_ON_OFF	names of testlets, variants, test cases, assesslets, requirements and file names in examples
"Throttle Concurrent Builds"	places where you can find something in external programs; names of text fields, sections, columns, menu items, buttons, or checkboxes in external programs
[Formatting]	placeholder for an element in TPT that has no name but should be explained

The following table describes the attention markers used in this documentation.

	This icon indicates a tip.
	This icon indicates a warning or restriction.

## 2 Introduction

TPT can be run in a Docker container to execute tests. You are free to split the tests using a TPT API script and run them in several containers of the same image.

You will need:

- Docker (<https://docs.docker.com/>)
- a headless TPT for Linux-based Docker images, that is a TPT that can only be executed via the command line
- a TPT project file

A license is required to execute TPT tests with Docker. Adjust the `license_default.cfg` in the TPT installation according to your license server data. The license server version must be at least 11.18.0.

```
# Example of a license_default.cfg for the TPT License Server
IP-Adr=lic.piketec.com
proxyType=<None>
port=30551
user=maxmustermann
password=mu34ma00a
LicenseType=TPTLicenseServer
proxyPort=0

# Example of a license_default.cfg for Flex LM
IP-Adr=30551@mylicenseserver.com
LicenseType=FlexLm
```

Docker cannot be used with a TPT Dongle license.



At the moment, tests on Linux can only be run on the EXE platform in a Docker container. Windows based images support at least the EXE platform and C/C++ platform.

To help you get started, the TPT installation folder includes several sample files for Linux-based and Windows-based Docker images that you can adapt to your needs or simply use as inspiration.

Unpack the Docker package which includes the following folders:

Folder	Content
0_Docker_Image	<ul style="list-style-type: none"> <li>• headless TPT installation for Linux-based Docker images</li> <li>• <code>Dockerfile</code>: contains all commands to build a Docker image</li> <li>• <code>build.bat</code>: batch script to build the Docker image</li> <li>• <code>build.sh</code>: shell script to build the Docker image</li> </ul>
1_SUT	<ul style="list-style-type: none"> <li>• example SUT</li> <li>• <code>make.sh</code>: makes the SUT executable in the Docker image using dynamic links</li> <li>• <code>make_static.sh</code>: makes the SUT executable in the Docker image using static links</li> <li>• <code>build_sut_in_docker.bat</code>: batch script to create a Docker container based on the Docker image and runs the shell script to make the SUT executable; removes the container</li> <li>• <code>build_sut_in_docker.sh</code>: shell script to create a Docker container based on the Docker image and runs the shell script to make the SUT executable; removes the container</li> </ul>
1_TPT_Example	TPT project file with the test cases to be executed
2_Run_Simple_SUT_w_CLI	<code>run_simple_SUT_w_CLI.bat</code> : batch script to start a Docker container based on the Docker image, mounts external volumes to the TPT project file, to a test result directory, and to the executable SUT; runs the test cases of the TPT project file and stores the test results in the results directory, and removes the container

Folder	Content
	<p><code>run_simple_SUT_w_CLI.sh</code>: shell script to start a Docker container, s. <code>run_simple_SUT_w_CLI.bat</code></p>
3_Run_1_Instance_with_API	<ul style="list-style-type: none"> <li>• <code>docker-compose.yml</code>: creates a Docker container based on the Docker image, mounts the needed external volumes, sets the needed ports for the API commands, starts TPT.</li> <li>• <code>run_1_instance_with_API_compose.bat</code>: executes the <code>docker-compose.yml</code></li> <li>• <code>run_1_instance_with_API.bat</code>: does the same as <code>docker-compose.yml</code> but must be executed in a command line</li> <li>• <code>talk_to_docker.tptapi</code>: API script with the commands necessary to run tests in TPT via the TPT API.</li> </ul>
4_Run_3_Instances_with_API	<ul style="list-style-type: none"> <li>• <code>docker-compose.yml</code>: creates three Docker containers based on the Docker image, mounts the needed external volumes, sets the needed ports for the API commands, starts TPT.</li> <li>• <code>run_3_instances_with_API_compose.bat</code>: executes the <code>docker-compose.yml</code></li> <li>• <code>run_3_instances_with_API.bat</code>: does the same as <code>docker-compose.yml</code> but must be executed in a command line</li> <li>• <code>talk_to_docker.tptapi</code>: API script with the commands necessary to run tests in TPT via the TPT API.</li> </ul>

Folder	Content
<p>Docker_C_Platform</p> <p>(only for Windows available)</p>	<ul style="list-style-type: none"> <li>• <b>Dockerfile:</b> contains all commands to build a Windows-based Docker image</li> <li>• <b>build_testframe.bat:</b> starts TPT and uses the API script <code>recompile.tptapi</code> in the folder <code>Scripts</code> to build the test frame using the C platform</li> <li>• <b>execute_testcases.bat:</b> executes the test cases from the example available in the folder <code>C-Example</code></li> </ul>



# 3 TPT Docker image

A Docker image contains the code that is needed to execute software in a Docker container.

PikeTec offers a headless TPT version that is needed to build a Docker image based on Ubuntu (Linux). To build the Ubuntu-based TPT Docker image, navigate to the directory of your headless TPT version and place the Docker file next to the folder of the headless TPT version.

To create a Windows based Docker image, create two new folders in `0_Docker_Image_Windows`. Name one of it `MinGW` and copy your MinGW installation into this folder. Name the other folder `TPT_installation` and copy your TPT installation into it.

## Example

Example folder: `<headless TPT installation>\0_Docker_Image`

The folder `0_Docker_Container_Image` contains a Docker file that includes all commands needed to build a Docker image. Run the build script to execute the Docker file and to build the image.

To create a Windows based Docker image add a MinGW-folder name `MinGW` and a copy of a TPT installation in the folder `TPT`.

# 4 Executable of SUT

To run the SUT in Docker, you have to modify the executable file. You need

- the C code to be tested
- a shell script to build the executable inside the Docker container
- and a batch file that creates the Docker container based on a TPT Docker image with the generated executable SUT and generates the test drivers

For the Linux-based Docker image, a C compiler is installed via Docker package manager, therefore the scripts can use gcc without changing anything. In the Windows-based image, the path to the gcc needs to be modified to make the gcc command available.

After the test driver generation is finished, the container can be terminated, thus removed.

## Example

Example folder: `<headless TPT installation>\1_SUT_Linux`

Run the batch file `build_sut_in_docker.bat` to create a Docker container based on the Docker image `tpt_headless_base` and to run the shell script to create an executable file that can be run in Linux. The container will be automatically removed after its task is finished.

# 5 TPT Docker container

A Docker container is a temporarily running instance of a Docker image. When a Docker container is closed, it is reset to the state of the underlying image. Any changes made during the execution are discarded. So, to keep test results even when the container is not active, you need to mount external volumes.

You must specify for each Docker container which TPT project has to be executed and which execution configuration should be used. The test cases to be executed are specified in the execution configuration.

You can execute tests in a Docker container by using an API script or a batch script.

## Mount volumes

You should always mount at least the following volumes in your Docker container:

- volume to your SUT
- volume to your TPT project file
- volume to the test results storage

In a batch file, the syntax for mounting the volume to the SUT is as follows:

```
--mount type=bind,src=<path to the folder with the SUT in  
Windows>,dst=<path to a specific folder in Linux>
```

For example:

```
--mount type=bind,src=C:\Tools\Examples\Docker\1_SUT_  
Linux,dst=/temp/tptdata/sut
```

In a Docker compose file, this might look like this:

```
volumes:  
  - type: bind  
    source: ${C:\Tools\Examples\Docker\1_SUT_Linux}  
    target: /temp/tptdata/sut
```

More about the syntax of Docker files, see <https://docs.docker.com/engine/reference/builder/>

## 5.1 Running tests in a Docker container via command line

To run tests via the command line, you must specify which project and which execution configuration needs to be run. Add the command to the same batch file that you use to create the Docker container.

Such a batch file might look like this:

```
:: create the Docker container and name it "tpt1"
docker run -it --name tpt1^

:: mount external volumes
--mount type=bind,src=C:\Tools\Examples\Docker\1_SUT_Linux,dst=/temp/tptdata/tpt_prj^
--mount type=bind,src=C:\Tools\Examples\Docker,dst=/temp/tptdata/results^
--mount type=bind,src=C:\Tools\Examples\Docker\1_TPT_Example,dst=/temp/tptdata/sut^
.
:: launch TPT Docker image in Docker container
tpt_headless_base^
.
:: run TPT headless, load the project file "LightControl.tpt", execute the
execution configuration "exeConfig", and store the results
/tpt/tpt_linux --run build --headless .
/temp/tptdata/tpt_prj/LightsControl.tpt.exeConfig --dataDir
/temp/tptdata/results

:: delete docker file "tpt1"
docker rm tpt1
```

Figure 5-1: Example of a batch file for creating Docker container

### Example

Example folder: <headless TPT installation>\2\_Run\_Simple\_SUT\_w\_CLI

Run the batch file `run_simple_SUT_w_CLI.bat` to create a Docker container based on the Docker image. The external volumes are mounted, so the Docker container has access to the TPT project file, to the executable SUT, and to a test result directory. The test cases of the execution configuration named **exeConfig** are executed with TPT. After the test results have been stored in the specified directory, the container will be removed.

## 5.2 Running TPT in a Docker container via API server

You can communicate with the TPT that is running in the Docker container from outside by using TPT-specific API commands. To do this, you must run the API server in the TPT Docker container and specify a port on which to pass the API commands from outside. You need to map the network port of your physical network card to a port of Docker's internal network, for example. `-p 1100:1099`, meaning `[host_port]:[docker_internal_image_port]`.

In addition, you must define an answer port for the communication. With the environment variable `TPT_RMI_PORT`, you can specify the answer port to be used by TPT, for example `--env TPT_RMI_PORT=40243`. The port must also be mapped to an open port via `-p`, for example `-p 40243:40243 --env TPT_RMI_PORT=40243`.

To load the TPT API server, start TPT with the following arguments: `--apiPort [docker_image_port_for_TPT_API] --apiBindingName TptApi --run apiserver --headless`.


The `apiPort` and `apiBindingName` can also be specified in the `apiserver.xml` file (see headless TPT folder: `0_Docker_Container/TPT/tpt.config.dir/apiserver.xml`). When you start the API server via the command line with an `apiPort` and an `apiBindingName` that differs from the specification in your `apiserver.xml` file, the command line specification wins.

### Send API commands via an API script to a Docker container

To pass TPT API commands to a TPT in a Docker container, you can either use a custom Java program or an API script. You can create and maintain this script in a normal TPT using the API Script Editor, or use a text editor and save the file as `*.tptapi`.

It is essential to specify the following in the API script file:

- directory to the TPT API script
- a host variable, for example `HOST = "localhost"`
- a binding, for example `BINDING = "TptApi"`; the binding name is specified when you start the Docker container using the argument `--apiBindingName`

To send the API commands to TPT, open the API script in a normal TPT, adjust the path to the TPT project you like to run. Then, click  **Run**.

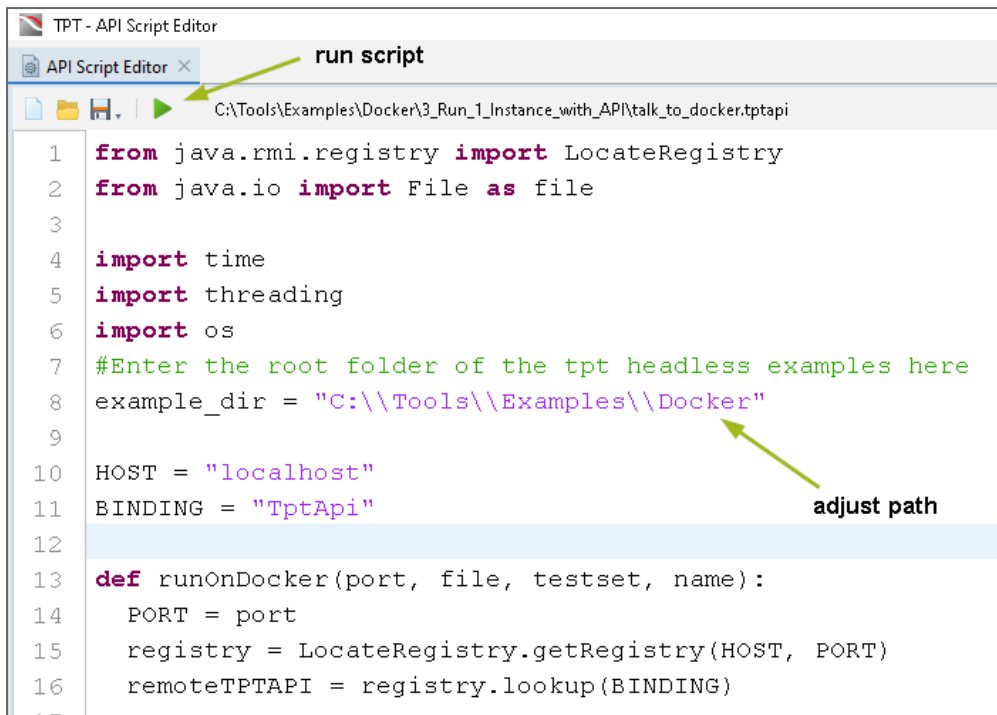


Figure 5-2: API script in the 'API Script Editor' of TPT

You can also run API scripts directly in the TPT Docker container, for example: `/tpt/tpt_linux --run apiserver my_api_script.tptapi --headless`. The `apiPort` and `apiBindingName` can be omitted in this case, since the communication takes place within the TPT Docker container and the API script terminates automatically when it is finished.

#### Example: Run test in one Docker container with API

Example folder: `<headless TPT installation>\3_Run_1_Instance_with_API`

Check the volumes to be mounted and the port forwarding specified in the `docker-compose.yml` and the `run_1_instance_with_API.bat`.

Run the batch file `run_1_instance_with_API_compose.bat` to execute the services specified in the YAML file `docker-compose.yml`. A Docker image will be created based on the existing Docker image `tpt_headless_base`, the necessary external volumes are mounted, the ports are forwarded, and the TPT headless image is started in the Docker container.

Instead of `run_1_instance_with_API_compose.bat`, you can run `run_1_instance_with_API.bat`. The execution of this file leads to the same results as the execution of the `run_1_instance_with_API_compose.bat` but does not make use of the `docker-compose.yml`.

### Example: Run test in one Docker container with API (continued)

When the Docker container is running, open the TPTAPI script file `talk_to_docker.tptapi` in the API Script Editor in the TPT user interface, and run the API script.

If you want to distribute test cases to multiple Docker containers, you only need to create multiple Docker containers based on the TPT Docker image.

To communicate with the containers via the TPT API, set a different port share and port forwarding for each container. Adjust the value of the `TPT_RMI_PORT` environment variable accordingly, for example:

```
docker run -dit --network=bridge --name tpt1 -p 1100:1099 -p
40243:40243 --env TPT_RMI_PORT=40243
[...]
docker run -dit --network=bridge --name tpt2 -p 1101:1099 -p
40244:40244 --env TPT_RMI_PORT=40244
[...]
docker run -dit --network=bridge --name tpt3 -p 1102:1099 -p
40245:40245 --env TPT_RMI_PORT=40245
```

Make sure to use different test results directories for each Docker container. A test report will be generated for each Docker container.

You can specify in the API script that certain test sets are executed in specific Docker containers, but you can also split test cases of a test set numerically. For more information about the TPT API, see [TPT API](#).

### Example: Run test in three Docker containers with API

Example folder: `<headless TPT installation>\4_Run_3_Instances_with_API`

Check the ports of all three Docker containers and add a different results path to each of them. The execution is as described in [Example: Run test in one Docker container with API](#)