PTR (Parallel Test Runner)

PTR has been developed to revolutionise the execution of test cases. It can reduce your test case execution time by up to 95% by running them in parallel. And much more than just parallel execution of test cases.

Contents

Overview

Automation testing (unit test cases, integrated test cases and UI automated test cases) is becoming more and more prevalent day by day. It's not unusual anymore for a project to have thousands of automated test cases. However, if the time the automated test cases of a project take to execute, exceeds say 30 minutes, it can wreak havoc with team's productivity. So to execute automated test cases quickly, they must execute in parallel and that is what PTR can help you in.

Parallel test case execution is just one thing; many times apart from running test cases in parallel, developers need special features that can minimise the manual intervention in running test cases with different configurations. Keeping this in mind, PTR has been developed to revolutionise the execution of your test cases. Here are the features it offers:

1. PTR can be used to run your test cases in parallel. Depending on the resources of your system like RAM and the number of cores it has, PTR can reduce the execution time by up to 95%. Isn't it revolutionary fast!

2. PTR can be used to run test cases written with most popular .net testing frameworks viz. NUnit, MSTest, CodedUI and SpecFlow. Soon it will also give support to run test cases written with testing frameworks like XUnit, JUnit, Cucumber, TestNG, etc. and with almost all popular languages like Python, Perl, Ruby, JavaScript etc. So far PTR is a Windows product but in next few releases, it will have support to run test cases on Mac, Linux, Ubuntu etc.

3. PTR automatically merges test result files and so you don't need any special handling of intermediate result files.

4. PTR can dynamically change the configurations of your testing project and that too at different phases of your test-run. It reduces the manual intervention required to run same test cases but with different configurations.

5. PTR can automatically rerun the failed test cases. While running failed test cases, it can also change the configuration of your testing project. Just imagine, you want to run your test cases on an iPhone but if some of them fail, you want to rerun the failed ones on an Android device without manual intervention. PTR can easily handle such situations.

6. A single instance of PTR can be used to run test cases of many different testing projects by having separate test-configuration of each project. Also, it can be used to run test cases of the same testing project with different configurations in parallel.

7. PTR allows you to not only configure the number of parallel threads but also to configure the maximum number of total threads. For example, you want to run test cases of two different test-configurations each with the thread count of 5 (so total number of threads will be 10). But your system's resources will perform best with only 7 as the maximum number of threads running at the same time. You can set the max thread size as 7 and PTR will not let the thread count to exceed 7 anytime. As soon as a thread completes its job and terminate, PTR will create another one and assign it the next job in the queue. So doesn't it offer the best utilisation of your resources!

8 PTR can be used to run test cases in parallel at both the level viz. class level and test case level. Class level means test cases of the same class will run sequentially but test cases of different classes will run in parallel. Test case level means, all the test cases can run in parallel irrespective of whichever class they belong to. PTR allows you to choose the concurrent unit (Test class or test case) as per your requirements.

9 PTR allows you to run your test cases with shared resources. Imagine you need to run your test cases on android devices. To run your test cases in parallel you buy 10 android devices (could be at your own premises or on the cloud). Now, these 10 devices are the shared resources that will be shared by your test cases during their execution. Once a device is allotted to a test case for its execution, that device could not be allotted to any other test case till current one finishes its execution. PTR can easily handle a situation like this by allocating and deallocating shared resources (android devices in this case) to the test cases at run-time only.

10 PTR can be used to run UI automated test cases on multiple machines via Selenium Grid.

11 PTR allows you to override existing test configurations by passing command line parameters to it. This feature could be of immense help while integrating PTR with CI tools like Jenkins, TFS, and TeamCity etc.

Getting started with PTR

PTR is very easy to use. You can download it from https://github.com/ParallelTestRunner/PTR.git. PTR uses **PTR.exe.config** to run your test cases. Open **PTR.exe.config** in any text editor like Notepad, Visual Studio or Notepad++. You will find three sections in it as below:

- **ProcessWideConfig**: This section contains a set of attributes that are applicable to the entire PTR process. In this section, you can specify various attributes such as your testing project, the path where you want to copy the result file, number of threads that will run test cases in parallel etc. Most of the attributes that you specify in this section can be overridden in "**Config**" nodes in "**TestingConfigurations"** section. If you want to apply the same attribute values to all "Config" nodes, it's better to specify them in "ProcessWideConfig" section as otherwise, you would have to specify those attributes in all "Config" nodes.

- **TestingConfigurations**: PTR has the concept of test-configuration. A test configuration specifies your testing project, test cases want to execute, a number of threads you want to create for running your test cases in parallel etc. A test-configuration is represented by a "**Config**" node in TestingConfigurations. You can create as many test-configurations as you want in single **PTR.exe.config** and a single PTR process will run all your test-configurations. Please note that most of the attributes in "ProcessWideConfig" section and in "Config" nodes are common. The attributes which you specify in a "Config" node, will override the values of same attributes in "ProcessWideConfig" section.

- **SharedResources:** This is the section where you specify shared resources of test cases. For example, you have 10 android devices and want to run your test cases in parallel on those devices. This is the place where you will specify those devices as shared resources. PTR will automatically allot those devices to test cases at runtime in such a way that no device is allocated to more than one test case at a time.

Here are the steps to run your test cases in parallel:

1. Download the PTR application from https://github.com/ParallelTestRunner/PTR.git.

2. Open **PTR.exe.config** in a text editor like Notepad, Visual Studio or Notepad++.

3. Specify the required attribute values in **"ProcessWideConfig"** and "**Config**" nodes in "TestingConfigurations" section. Please remember that you can create as many "**Config**" nodes as you want. If your testing project is written with NUnit, make sure that you have **nunit3-console.exe** in the path environment variable of your system. If your testing project is written with MSTest, make sure that you have **MSTest.exe** in the path environment variable of your system.

4. Just run the PTR.exe and it will run your test cases as per the attribute values you specified in above step.

Features:

There are various attributes in "**ProcessWideConfig**" section and in "**Config**" nodes in "**TestingConfigurations**" section that you can use to run your test cases as per your requirements. Most of the attributes are common in both the sections however some are specific to one section only.

LoggingLocation: PTR creates logs during the execution of test cases. These logs provide vital information that can help in troubleshooting and debugging. In this attribute, you specify the path of the directory where you want to create log file. Please note that this attribute could be set only in "ProcessWideConfig" section and not in "Config" nodes in "TestingConfigurations" section.

MaxThreads: The number of test cases PTR can run in parallel is same as the number of threads it can create. You can specify the maximum number of threads PTR is allowed to create with this attribute. The default value of this attribute is the number of logical processor/cores your system has (https://msdn.microsoft.com/en-us/library/windows/desktop/dd405503(v=vs.85).aspx). Please note that this attribute could be set only in "ProcessWideConfig" section and not in "Config" nodes in "TestingConfigurations" section.

How to know the most appropriate value of this attribute: The most appropriate value of this attribute depends on the number of logical cores your system has and on the behaviour of your test cases. If your test cases are mostly CPU bound, then probably you don't even need to set this attribute at all as PTR is smart enough to set the most appropriate value of this flag. But if your test cases are IO bound or a mix of both CPU & IO bound or UI automated ones (like the test cases that run on browsers and mobile devices) or you don't know the behaviour of your test cases at all, then you should assign it the value that comes out from the expression [**the number of physical cores your system has * 2**]. After that note down the total time that PTR takes to execute your test cases and keep on increasing the value of this attribute by 1 until you get your test cases executed faster than with the previous value of this attribute. After few attempts, you will get a most appropriate value that will make your test cases executed in the least possible time.

Note: The value you assign to this attribute is also the most appropriate value for "ThreadCount" attribute in all test-configurations as well (Read ThreadCount).

ThreadCount: This attribute specifies the number of test cases you want to run in parallel. The default value of this attribute is the number of logical processor/cores your system has (https://msdn.microsoft.com/en-us/library/windows/desktop/dd405503(v=vs.85).aspx). You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

How to know the most appropriate value of this attribute: The most appropriate value of this attribute in each test-configuration (i.e. in each "Config" node in "TestingConfigurations") is same as of "MaxThreads" attribute.

TestingProjectLocation: Here you specify either the path of the directory which contains your testing projects or path of a file which contains paths of your test assemblies.

Here are the guidelines for this attribute:

1. If you specify the path of a directory here, paths of the assemblies having test cases (that you specify in "SemicolonSeparatedFilesHavingTestCases") must be relative to this directory. For example below is how PTR.exe.config will look like in this case:

```
<Config TestingProjectLocation="C:\TestingProjectDirectory"
        SemicolonSeparatedFilesHavingTestCases="TestProject_1\debug\UnitTestAssembly.dll;TestProject_2\debug\BrowserTestCaseAssembly.dll"
        ●
        ●
        ●
```

   As you can see in above image, we have specified two assemblies having test cases (UnitTestAssembly.dll and BrowserTestCaseAssembly.dll) in "SemicolonSeparatedFilesHavingTestCases" and their paths are relative to that of "C:\TestingProjectDirectory" (as specified in "TestingProjectLocation" attribute)

2. If you specify the path of a file having paths of test assemblies in below format, you must keep "SemicolonSeparatedFilesHavingTestCases" as empty.

TestAssemblies.xml

```
<?xml version="1.0" encoding="utf-8"?>
<TestAssemblies>
  <TestAssembly Path="C:\TestingProjectDirectory\TestProject_1\debug\UnitTestAssembly.dll" />
  <TestAssembly Path="C:\TestingProjectDirectory\TestProject_2\debug\BrowserTestCaseAssembly.dll" />
</TestAssemblies>
```

   For example, let say that you create a file "C:\TestAssemblies.xml" with content as shown in above image, below is how PTR.exe.config will look like:

```
<Config TestingProjectLocation="C:\TestAssemblies.xml"
        SemicolonSeparatedFilesHavingTestCases=""
        ●
        ●
        ●
```

3. If you specify the path of a directory here but keep "SemicolonSeparatedFilesHavingTestCases" as empty, PTR will search all the assemblies having test cases and execute test cases of those assemblies.

   For example below is how PTR.exe.config will look like in this case:

```
<Config TestingProjectLocation="C:\TestingProjectDirectory"
        SemicolonSeparatedFilesHavingTestCases=""
        ●
        ●
        ●
```

So with above configuration, PTR will find all the assemblies having test cases, in "TestingProjectDirectory" directory (and its subdirectories) and execute them.

You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

SemicolonSeparatedFilesHavingTestCases: Here you specify the assemblies which contain your test cases. You can specify multiple assemblies by separating them with semicolons. If you keep this attribute empty, PTR will search for all the assemblies having test cases in the directory as specified in "TestingProjectLocation" (and its sub-directories). You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

Note: The paths of the assemblies having test cases that you specify here, should either be relative to the path of "TestingProjectLocation" or be the full paths.

ExecutionLocation: Here you specify the path of the directory where you want to run your test cases. This is the place where PTR will create all its intermediate result files and other temporary data. If you don't specify this attribute or keep its value empty, PTR will consider its own parent directory as the execution location. You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

TestRunner: This is a mandatory attribute. Here you specify test runner that will actually trigger the test cases. The test runner is usually a bat file and here are the possible values of this attribute:

- **Nunit3TestRunner.bat**: When you download PTR, you will find a bat file named "Nunit3TestRunner.bat". This file runs NUnit test cases (and the SpecFlow test cases with NUnit as the testing framework) using **nunit3-console.exe** (which is a NUnit test case runner, can be downloaded from https://github.com/nunit/nunit-console/releases/tag/3.6). Please note that for using this file you must have **nunit3-console.exe** in the path environment variable of your system.

- **Nunit2TestRunner.bat:** When you download PTR, you will find a bat file named "Nunit2TestRunner.bat". This file runs NUnit test cases (and the SpecFlow test cases with NUnit as the testing framework) using **nunit3-console.exe** (which is a NUnit test case runner, can be downloaded from https://github.com/nunit/nunit-console/releases/tag/3.6) but creates the result file with the format as of NUnit version 2. Please note that for using this file you must have **nunit3-console.exe** in the path environment variable of your system.

- **MSTestRunner.bat:** When you download PTR, you will find a bat file named "MsTestRunner.bat". This file runs your test cases written with MSTest and CodedUI and the ones that are written with SpecFlow with MSTest as the testing framework. Please note that for using this file you must have **MSTest.exe** in the path environment variable of your system.

You can specify "TestRunner" attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

TestCasesExtractor: This is a mandatory attribute. This attribute is used to specify the executable file which can extract test cases from an assembly. Here you can use a single value "**DotNetTestCasesExtractor.exe**" for extracting test cases written with any .net testing framework like NUnit, MSTest, CodedUI or SpecFlow. In future, there would be multiple possible values to extract test cases written with other testing frameworks. You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

ReportProcessor: This is a mandatory attribute. This attribute is used to specify the executable file which can process test-result files. Here are its possible values:

- **NUnitReportUtility.exe:** This file is downloaded with PTR. If your test cases are written with NUnit as the testing framework (or SpecFlow with NUnit as the testing framework), specify this file as a value to the "ReportProcessor.

- **TrxUtil.exe:** This file is also downloaded with PTR. If your test cases are written with MSTest or CodedUI or SpecFlow with MSTest as the testing framework, specify this file as a value to the "ReportProcessor".

You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

ConcurrentUnit: This attribute is used to specify how you want to run your test cases in parallel. The possible values of this attribute are **1** and **2**. If you assign 1 to this attribute, PTR will run all the test cases of a test class sequentially only but the test cases of different classes will run in parallel. If you assign 2 to this attribute, PTR can run all test cases in parallel irrespective of whichever class they belong to. The default value of this attribute is 2. You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

CleanAfterCompletion: While running test cases, PTR creates intermediate temporary files and directories at the location as specified by "ExecutionLocation" attribute. As the name suggests, you can decide whether PTR should do the clean-up operation after the execution of test cases or not. The possible values for this attribute are true and false only. The default value of this attribute is true. You can specify this attribute in both "ProcessWideConfig" section and in "Config" nodes in "TestingConfigurations" section. If you specify this attribute in "Config" nodes, it will override the value specified in "ProcessWideConfig" section.

ID: This attribute is applicable only to "Config" nodes in "TestingConfigurations" section. This is used to uniquely identify PTR's test configurations. You must ensure that this attribute is unique in all "Config" nodes.

IsEnabled: This attribute is applicable only to "Config" nodes in "TestingConfigurations" section. As the name suggests, you can decide whether PTR should run a particular test-configuration or not. The possible values of this attribute are true and false only. The

default value of this attribute is true. If you assign false to this attribute, PTR will not run test cases as per that test configuration.

ReportingLocation: This attribute is applicable only to "Config" nodes in "TestingConfigurations" section. In this attribute, you specify the path of the result file that PTR will create after all the test cases are executed. You must ensure that you assign different file paths to this attribute in each "Config" node as otherwise the result file of one "Config" node can override the result file of other "Config" nodes. Also, you must provide a file path and not a directory here.

TestCategories: Using this attribute you can specify test categories you want to execute. You can specify multiple categories by separating them with operators '^' (AND), '|' (OR) and '!' (NOT). Please note that you can't use '^' and '|' operators together.

Examples:

- *TestCategories="Catgory_1"*: Will execute test cases that belong to the category "Catgory_1".
- *TestCategories="Catgory_1^Catgory_2"*: Will execute test cases that belong to both of the categories "Catgory_1" and "Catgory_2".
- *TestCategories="Catgory_1|Catgory_2"*: Will execute test cases that belong to either "Catgory_1" or "Catgory_2" or both.
- *TestCategories="Catgory_1^Catgory_2^!Catgory_3"*: Will execute test cases that belong to both the categories "Catgory_1" and "Catgory_2" but do not belong to the category "Category_3".
- *TestCategories="!Catgory_1^!Catgory_2^!Catgory_3"*: Will execute test cases that do not belong to any of the categories "Catgory_1", "Catgory_2" and "Category_3".

Note: All the test case filter attributes "TestCategories", "TestClasses", "SemicolonSeparatedTestCases" and "SemicolonSeparatedTestCasesToBeSkipped" are independent attributes and so any combination of them can be used in a test-configuration.

TestClasses: Using this attribute you can specify test classes you want to execute. You can specify multiple test classes by separating them with '|' (OR) operator. Please note that you can't use any other operator here.

Example:
TestClasses="Class_1|Class_2|Class_3": Will execute test cases that are defined in either "Class_1" or "Class_2" or "Class_3".

Note: All the test case filter attributes "TestCategories", "TestClasses", "SemicolonSeparatedTestCases" and "SemicolonSeparatedTestCasesToBeSkipped" are independent attributes and so any combination of them can be used in a test-configuration.

SemicolonSeparatedTestCases: Using this attribute, you can specify test cases you want to execute.

Example:
SemicolonSeparatedTestCases="TestCase_1;TestCase_2;TestCase_3": Will execute test cases with names "TestCase_1", "TestCase_2" and "TestCase_3".

Note: All the test case filter attributes "TestCategories", "TestClasses", "SemicolonSeparatedTestCases" and "SemicolonSeparatedTestCasesToBeSkipped" are independent attributes and so any combination of them can be used in a test-configuration.

SemicolonSeparatedTestCasesToBeSkipped: Using this attribute, you can specify test cases you don't want to execute.

Example:
SemicolonSeparatedTestCasesToBeSkipped="TestCase_1;TestCase_2;TestCase_3": Will execute test cases other than with names "TestCase_1", "TestCase_2" and "TestCase_3".

Note: All the test case filter attributes "TestCategories", "TestClasses", "SemicolonSeparatedTestCases" and "SemicolonSeparatedTestCasesToBeSkipped" are independent attributes and so any combination of them can be used in a test-configuration.

TimesToRerunFailedTestCases: Test cases sometimes fail because they get timed out, or the system is having less network bandwidth or any other reason that is does not reflect failure in the functionality they were written for. In such cases, it may make sense to rerun failed test cases. You can you this attribute to specify, how many times you want to run failed test cases. Please note that if you use this attribute to run failed test cases multiple times, PTR will take more time to finish the execution.

Advanced features:

MinBucketSize: A thread that executes test cases, pick as many test cases as the current bucket size is, in each of its iteration. This attribute ensures that current bucket size never goes below a certain number. For example, you are running 100 test cases with 4 threads. PTR will create 4 threads and allocate as many test cases to them as the current bucket size is. Let's say that current bucket size is 50. But PTR can't allocate 50 test cases to each thread (it can allocate maximum up to 100/4 i.e. 25 test cases to each thread). In that case PTR will reduce the current bucket size to half i.e. 25. But if the value of this attribute is say 30, PTR will not let the current bucket size to go below 30. So PTR will allocate 30 test cases to three threads and remaining 10 test cases to the fourth thread. **The default value of this attribute is 3.**

MaxBucketSize: A thread that executes test cases, pick as many test cases as the current bucket size is. When PTR starts the execution of test cases, the current bucket size is set as equal to the value of this attribute. But if and when, the remaining test cases count goes below [current bucket size * total thread count], the current bucket size get halved subject to the condition that it never goes below "MinBucketSize". **The default value of this attribute is 50.**

SemicolonSeparatedConfigResultsToBeMergedInto: If you want to merge the results the different test-configurations, you can use this attribute to do so. For example, let's say that you have two test-configurations (i.e. two Config nodes in TestingConfigurations section) and their ID attributes' values are "ID_1" and "ID_2". If you want to have a consolidated reports of both the test-configurations, you can set SemicolonSeparatedConfigResultsToBeMergedInto="ID_2" in test-configuration with ID="ID_1" and PTR will merge the results of test-configuration with ID="ID_2" into the one with ID="ID_1". But please note that PTR can merge results only if both the configurations produce results in same format. Like, you can't merge results of MsTest test cases with that of NUnit. Please note that this attribute is applicable only to "Config" nodes in "TestingConfigurations" section.

BeforeTestExecution: If you want to execute some other application before running PTR, you can specify the path of that application here. For example you may want to execute some executable that does some infrastructure setup required for executing your test cases. Please note that this attribute could be set only in "ProcessWideConfig" section and not in "Config" nodes in "TestingConfigurations" section.

AfterTestExecution: If you want to execute some other application after PTR finishes the execution of all the test cases, you can specify the path of that application here. For example you may want to specify some executable that does some clean-up operation after the execution of test cases. Please note that this attribute could be set only in "ProcessWideConfig" section and not in "Config" nodes in "TestingConfigurations" section.

MakeTestRunnerAsChildProcessOfPTR: Assigning true to this attribute will make your test runner (Nunit3TestRunner.bat, Nunit2TestRunner.bat and MsTestRunner.bat) to run in hidden mode and so less cluttering on your desktop. Also, if you close the PTR process explicitly, it will automatically close all the hidden test-runner processes as well. If you assign false to this attribute, your test runner will run in a separate command window and if you close the PTR process explicitly, it will not close test-runner processes automatically. Here are the guidelines for this attribute:

1. If your test cases do not run on a GUI (desktop applications and browsers), make this flag as true.
2. If your test cases run on mobile devices, make this flag as true.
3. If your test cases run on any device/browser on the cloud, make this flag as true.
4. If your test cases run on browsers but through selenium grid only, make this flag as true.
5. If your test cases run on browsers without selenium grid, make this flag as false.

WorkingDirectoryOfTestRunner: Sometimes a testing project seeks for files/directories relative to its working directory. But since you'll be running your testing project through PTR only, the working directory of PTR can become the working directory of your testing project (it all depends on the how your test runner is implemented). So your testing project may not get the correct paths of files/directories it needs to run its test cases. Although most of the test-runner (MsTest.exe, nunit3-console.exe) are smart enough to handle this and so no issue will appear while running your test cases, but in case you find such issues, you can use this attribute to set the working directory of your test runner.

LoadTestingProjectBinariesFromItsOwnLocationOnly: Using this option, you can tell PTR whether to copy your testing project binaries at execution location (the path

specified at ExecutionLocation) or not. **By default this flag would be false**, so all the binaries would be copied at execution location. But if the size of the binaries of your testing project is large say 200 MB or more, make this attribute as true to avoid unnecessary delay in copying such a large chunk of binaries.

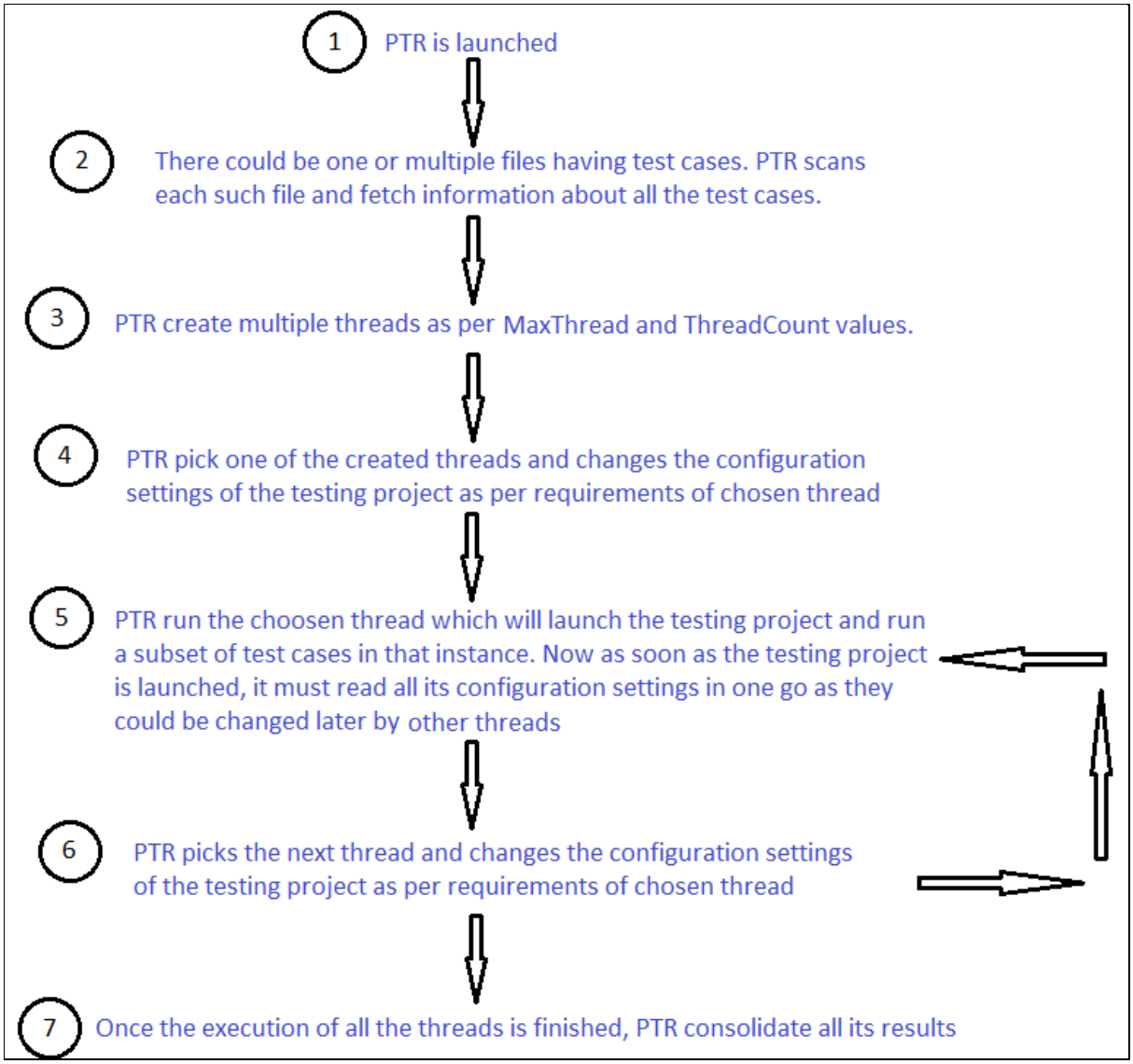
BeforeRunConfigEditor: Generally, every testing project has its configurations using which developers can change the behaviour of their test cases. For example, one can have a web automated testing project and its test cases will run on a browser depending on the browser settings (such as browser type - chrome, ie, firefox) in project's configuration file (like generally, in .net projects configurations are maintained in App.config file). Configurations are usually maintained in files such as .xml, .json, .properties and sometimes in registry too.

Using PTR's this attribute (BeforeRunConfigEditor), you can change the configuration settings of your testing project before running its test cases. But why would you need that? Read below scenarios:

- Just imagine that you want to run your web automated test cases on three browsers chrome, ie and firefox in parallel. But at a time you can configure your testing project to run only on one browser. So how would you run your test cases on all three browsers? This is where PTR can help you in. You will create three test-configurations (i.e. three Config nodes in TestingConfigurations in PTR.exe.confg) and in each test-configuration, you will use *BeforeRunConfigEditor* attribute to change the browser type you want in your testing project's configuration. And then when you will run PTR, it will run your testing projects in all three browsers at same time.
- Imagine you have a testing project whose test cases run on a mobile device. You have bought 10 mobile devices on cloud (like Perfecto, Sauce lab etc.) and to get faster results you want to run your test cases in parallel on those devices. You would need to create 10 different threads and distribute all your test cases to those threads. Each thread then would be allocated a mobile device and no other thread can have access to the mobile devices allocated to other threads. This way you would be running 10 test cases in parallel on 10 mobile devices. All this thread creation and allocation of mobile devices should be dynamic. PTR can easily make you achieve this using *BeforeRunConfigEditor* and *SharedResources*.

How to use BeforeRunConfigEditor?

Prerequisite: When your testing project is loaded, the very first thing it should do is read all its configuration settings before running its test cases. Why? PTR will launch multiple instances of your testing project and each instance of your testing project may intend to run with different configuration settings. Let's assume that thread T1 changes configuration settings to CS1, and launch a testing project instance TP1. Then thread T2 changes configuration settings to CS2, and launch a testing project instance TP2. Now, if testing project instance TP1 read configuration settings, it would read CS2 and not the CS1 which means TP1 will run with wrong/unintended configuration settings. It may give you inconsistent result. To overcome this, a testing project instance must read all its configuration settings at one go before starting the execution of its test cases assigned to it. PTR put sufficient delay between two testing project instances so that every instance read the correct and intended configuration settings only.



Using BeforeRunConfigEditor to change configuration settings: You must assign this attribute some executable file that will change the configuration settings of your testing project. PTR comes up with one such executable "XmlEditor.exe". If your configuration settings are in some xml file, you can use XmlEditor.exe to change your configuration settings. Click here to see how XmlEditor.exe works.

Here is a typical example of the value that could be assigned to BeforeRunConfigEditor:

```
BeforeRunConfigEditor="     XmlEditor.exe
                            @Param/Path:C:\Users\aseem\Desktop\XmlDoc.xml
                            @Param/Path:C:\Users\aseem\Desktop\XPaths.txt"
```

As mentioned above, you must specify some executable file that will change your testing project's configurations. In above example, that executable file is "XmlEditor.exe". After the executable file path, you can specify the command line parameters that you want to send to that executable file. For example, the first command line parameter that XmlEditor.exe takes is the xml document that you want to change and the second parameter is the .txt file that contains the XPaths that you want to change.

With above value of BeforeRunConfigEditor, PTR will run XmlEditor as below:

`XmlEditor.exe "C:\Users\aseem\Desktop\XmlDoc.xml" "C:\Users\aseem\Desktop\XPaths.txt"`
And XmlEditor will change XmlDoc.xml file as per XPaths specified in XPath.txt file.

Note: In above example, you can see that we are passing command line parameters by prefixing them with the string "`@Param/Path:`". Any string that we want to send as a command line parameter to the executable, must start with "@Param" and if that parameter is some path of a file, it must starts with "/Path". Also if the files' paths that you pass as command line parameters are relative to either you're testing project (specified by "**TestingProjectLocation**" attribute) or PTR's parent directory, you don't need to specify the full path of the file. For example, if your testing project has a file say app.exe.config in the directory of your testing project, you can pass it as a command line parameter without specifying its full path; it would be passed as "@Param/Path:app.exe.config".

If you have some shared resources (for example you have 10 mobile devices and you want to run your test cases on all those devices), you can specify your resources in `<SharedResources>` section and use them with BeforeRunConfigEditor (Click here to know how to use shared resources).

BeforeRerunConfigEditor: This attribute is similar to BeforeRunConfigEditor with only one difference. If you want to run failed test cases automatically but with different configuration settings, you can use BeforeRerunConfigEditor for that. BeforeRerunConfigEditor will work same way as BeforeRunConfigEditor does but only when PTR rerun the failed test cases i.e. it will change the configuration settings just before rerunning failed test cases.

How to use XmlEditor.exe? PTR comes up with one application called "XmlEditor.exe". XmlEditor.exe can be used to change the values of nodes and attributes in an xml file. It takes following command line parameters:

1. The first argument is the path of the xml file you want to edit.
2. The second argument is the .txt file that contains XPaths along with values to be assigned to the elements evaluated with those XPaths. Let's assume that you want to modify below xml file (its path would be provided as the first command line parameter to XmlEditor.exe):

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <ProcessWideConfig    MaxThreads="5"
                        SemicolonSeparatedFilesHavingTestCases="xyz1.dll;xyz2.dll" />
  <TestingConfigurations>
    <Config ID="SmokeTestcases"
            IsEnabled="false"
            TestingProjectLocation="C:\CalculatorTestingProject\bin\Debug"
            ThreadCount="9"
            TimesToRerunFailedTestCases="0"
            CleanAfterCompletion="false" />

    <Config ID="RegressionTestcases"
            IsEnabled="false"
            TestingProjectLocation="C:\CalculatorTestingProject\bin\Debug"
            ThreadCount="8"
            TimesToRerunFailedTestCases="0"
            CleanAfterCompletion="false" />
  </TestingConfigurations>
</configuration>
```

Below is an example of the file that contains XPaths along with values one may want to assign to respective elements (its path would be provided as the second command line parameter to XmlEditor.exe):

```
ProcessWideConfig/@MaxThreads:=:8
TestingConfigurations/Config[@ID="SmokeTestcases"]/@IsEnabled:=:true
TestingConfigurations/Config[@ID="RegressionTestcases"]/@IsEnabled:=:true
TestingConfigurations/Config[1]/@ThreadCount:=:5
TestingConfigurations/Config[2]/@ThreadCount:=:5
TestingConfigurations/Config[1]/@TestingProjectLocation:=:C:\CalculatorTestingProject\bin\Release
TestingConfigurations/Config[2]/@TestingProjectLocation:=:"C:\CalculatorTestingProject\bin\Release"

TestingConfigurations/Config[1]/@TimesToRerunFailedTestCases:=:%VAL_1%
TestingConfigurations/Config[2]/@TimesToRerunFailedTestCases:=:"%VAL_2%"

#TestingConfigurations/Config[1]/@CleanAfterCompletion:=:true
```

So if you pass the above xml file and txt file as first and second command line parameters to XmlEditor.exe, XmlEditor.exe will modify the xml file as below:

```
<?xml version="1.0" encoding="utf-8"?>
<configuration>
  <ProcessWideConfig     MaxThreads="8"
                         SemicolonSeparatedFilesHavingTestCases="xyz1.dll;xyz2.dll" />
  <TestingConfigurations>
    <Config ID="SmokeTestcases"
            IsEnabled="true"
            TestingProjectLocation="C:\CalculatorTestingProject\bin\Release"
            ThreadCount="5"
            TimesToRerunFailedTestCases="0"
            CleanAfterCompletion="false" />

    <Config ID="RegressionTestcases"
            IsEnabled="true"
            TestingProjectLocation="C:\CalculatorTestingProject\bin\Release"
            ThreadCount="5"
            TimesToRerunFailedTestCases="0"
            CleanAfterCompletion="false" />
  </TestingConfigurations>
</configuration>
```

The elements that would be modified by XmlEditor.exe are underlined in above image.

3. In the above .txt file you could see below XPaths:

- TestingConfigurations/Config[1]/@TimesToRerunFailedTestCases:=:%VAL_1%
- TestingConfigurations/Config[2]/@TimesToRerunFailedTestCases:=:"%VAL_2%".

In above XPaths, the value fields are enclosed with the percentage sign '%'. This means that the values to the elements corresponding to these XPaths would not be provided in the XPath file. Instead they would be provided by third or later command line parameters to XmlEditor.exe. For example, if you pass **VAL_1:=:2** and **VAL_2:=:3** as third and fourth command line parameters to XmlEditor.exe, it would assign 2 and 3 to TimesToRerunFailedTestCases attributes in first and second <Config> nodes in the .xml file.

Here are the rules of creating a correct XPath .txt file:

1. Values must be assigned to XPaths using **:=:** operator (and not = operator).
2. If you want to change the value of some attribute in an .xml file, the attribute name must start with **@** sign. For example, in XPath, "TestingConfigurations/Config[@ID="SmokeTestcases"]", the attribute ID starts with @ sign.
3. If you want to refer nth instance of an element in the XPath, it is not zero based. Any index in the XPath, starts with 1 and not 0.
4. The values that you want to assign to an element, can be provided with both double quotes ("") or without double quotes.
5. You can comment out an XPath using hash sign (#). This means, that XPath would not be considered while changing the .xml file.

SharedResources: Sometimes your test cases may have some shared resources like mobile devices, browsers on cloud, selenium grid, database server etc. If any of the shared resources is being accessed by a test case, you may not want other test cases to access that resource. For example, if you buy only one android device on cloud, you can run only one test case on it. That means you can run your test cases sequentially only. To run your test cases in parallel, you need to buy multiple devices on cloud. Let's say that for running 10 test cases in parallel, you buy 10 android devices. Now you need to run your test cases in a way that maximum ten test cases run at a time and each one is allocated an android device (out of 10 you bought on the cloud). No two or more test cases should access same device. And as soon as one test case completes, the device should be allocated to the next test case in the queue. All this should happen automatically and dynamically. PTR can help you to achieve this behaviour. So here is how to use **SharedResources** section to achieve this behaviour:

1. You can specify your shared resources as below.

   Exampl 1: There are 2 shared resources **VAL_1** and **VAL_2**. Both of them have two possible values; VAL_1 can have 11 & 22 and VAL_2 can have 33 & 44. Here is how they would be represented:

```
<SharedResources>
          <add ID="SR_1" SemicolonSeparatedResources="VAL_1:=:11"/>
          <add ID="SR_2" SemicolonSeparatedResources="VAL_1:=:22"/>
          <add ID="SR_3" SemicolonSeparatedResources="VAL_2:=:33"/>
          <add ID="SR_4" SemicolonSeparatedResources="VAL_2:=:44"/>
</SharedResources>
```

   Exampl 2: You can also group your resources as below:

```
<SharedResources>
          <add ID="SR_1" SemicolonSeparatedResources="VAL_1:=:11;VAL_2:=:33"/>
          <add ID="SR_2" SemicolonSeparatedResources="VAL_1:=:22";VAL_2:=:44"/>
</SharedResources>
```

   There are two combinations of VAL_1 and VAL_2 that would be applied to your testing project's configurations.

   Exampl 3: If you have two android devices; you can represent them as below:

```
<SharedResources>
          <add ID="SR_1"
SemicolonSeparatedResources="AndroidInstance:=:I1;Device:=:Samsung Note 6" />
          <add ID="SR_2"
SemicolonSeparatedResources="AndroidInstance:=:I2;Device:=:Samsung S7" />

</SharedResources>
```

2 You have to use "BeforeRunConfigEditor" or "BeforeRerunConfigEditor" attribute to change the configurations of your testing project. Below is how you can use shared resources in "BeforeRunConfigEditor":

For example 1:

```
BeforeRunConfigEditor="XmlEditor.exe
@Param/Path:C:\Users\aseem\Desktop\XmlDoc.xml
@Param/Path:C:\Users\aseem\Desktop\XPaths.txt
@Param/SharedResources:SR_1;SR_2 @Param/SharedResources:SR_4;SR_3"
```

For example 2:

```
BeforeRunConfigEditor="XmlEditor.exe
@Param/Path:C:\Users\aseem\Desktop\XmlDoc.xml
@Param/Path:C:\Users\aseem\Desktop\XPaths.txt
@Param/SharedResources:SR_1;SR_2"
```

For example 3:

```
BeforeRunConfigEditor="XmlEditor.exe
@Param/Path:C:\Users\aseem\Desktop\XmlDoc.xml
@Param/Path:C:\Users\aseem\Desktop\XPaths.txt
@Param/SharedResources:SR_1;SR_2"
```

3 Then you need to use your shared resources in XPath.txt as below:

```
ProcessWideConfig/@MaxThreads:=:8
TestingConfigurations/Config[@ID="SmokeTestcases"]/@IsEnabled:=:true
TestingConfigurations/Config[@ID="RegressionTestcases"]/@IsEnabled:=:true
TestingConfigurations/Config[1]/@ThreadCount:=:5
TestingConfigurations/Config[2]/@ThreadCount:=:5
TestingConfigurations/Config[1]/@TestingProjectLocation:=:C:\CalculatorTestingProject\bin\Release
TestingConfigurations/Config[2]/@TestingProjectLocation:=:"C:\CalculatorTestingProject\bin\Release"

TestingConfigurations/Config[1]/@TimesToRerunFailedTestCases:=:%VAL_1%
TestingConfigurations/Config[2]/@TimesToRerunFailedTestCases:=:"%VAL_2%"

#TestingConfigurations/Config[1]/@CleanAfterCompletion:=:true
```

4 Now when you would run PTR, it will choose the least used shared resource and send the value of "SemicolonSeparatedResources" attribute of the chosen resource to XmlEditor.exe. XmlEditor.exe would then fetch the values of "VAL_1" and "VAL_2" and apply to the respective XPaths (having %VAL_1% and %VAL_2% assignments).

Overriding ProcessWideConfig and TestingConfigurations/Config through command line parameters to PTR:

There can be situations when you want to change the values of various attributes of **ProcessWideConfig** and/or of "**Config**" nodes in TestingConfigurations but without opening **PTR.exe.config**. For example, you may want to change/override **IsEnabled** attribute in a particular testing configuration (i.e. a particular **configuration/TestingConfigurations/Config** node in PTR.exe.config) while integrating PTR with some continuous integration tool like Jenkins, TeamCity, Bamboo, TFS etc. PTR allows doing this via its command line parameters. Here is how to modify attributes of ProcessWideConfig and Config nodes using command line parameters:

1. The template of command line parameter to override an attribute of ProcessWideConfig is "**/ProcessWideConfig:AttributeName:Value**". For example to override "**MaxThreads**" attribute of ProcessWideConfig with value say 12, you will pass the command line parameter as below:

   /ProcessWideConfig:MaxThreads:12

2. The template of command line parameter to override an attribute of a specific test configuration i.e. "Config" node in "TestingConfigurations" is "**/ID:AttributeName:Value**" where ID is the value of ID attribute of "Config" node. For example to override the "**ThreadCount**" attribute of a specific "Config" node with ID equal to say ”SmokeTest” you will pass the command line parameter as below:

   /SmokeTest:TestRunner:6

3. The template of command line parameter to override an attribute of all "Config" nodes in "TestingConfigurations" is "**/TestingConfigurations:AttributeName:Value**". For example to override the "IsEnabled" attribute of all "Config" nodes in "TestingConfigurations", you will pass the command line parameter as below:

   /TestingConfigurations:IsEnabled:true