Using the Tachyon Policy Tool

Summary

How to use the Tachyon Policy Tool to import and manage Guaranteed State policy objects.The Policy Tool is avaliable in the **Tachyon Platform zip** file from the 1E support portal page (https://1eportal.force.com).

The Tachyon Policy Tool is a **command-line interface** tool used to import and manage the following Guaranteed State elements that are defined in XML files.

- Fragments
- Policies
- Rules
- Trigger templates

The tool is capable of performing the following operations

- Import and validate policy object definitions from one or more XML files
- Validate the policy object definitions only
- Show a summary list of policy objects of a given type on the server
- Delete a policy object with the given ID on the server
- Show policy objects referencing a given object instance specified by its ID
- Export general XML files from the policy object specified by its ID

On this page:

- Permissions
- Bulk importing data using the policy tool
- Command line syntax
  - Mandatory parameters
  - Action-specific options
- Examples
  - List all trigger templates
  - Export all fragments
  - Delete policy with Id=7
  - Show all references to fragment IDs 88, 89 and 90
- Source XML files
  - Fragments
  - Rules
  - Policies
  - Trigger Templates
- Updating an existing policy object
- Tachyon Platform zip

Permissions

To access the Guaranteed State application and its pages you must be a Tachyon user with at least one of the following permissions:

- **Guaranteed State Administrators** system role - has full control over the Guaranteed State configuration
- **Guaranteed State Viewers** system role - able to view the Guaranteed State configuration and reports
- Custom roles with **Guaranteed state** type permission - read, and write permission

Bulk importing data using the policy tool

A batch file is supplied with the policy pack which will import all components using the policy tool.

Currently it looks like this:

```
@echo off
Tachyon.Policy.exe -action=import -object=triggertemplates -file=TriggerTemplates\*.xml
pause
Tachyon.Policy.exe -action=import -object=fragments -file=Fragments\*-check-*.xml -fragmenttype=check
pause
Tachyon.Policy.exe -action=import -object=fragments -file=Fragments\*-fix-*.xml -fragmenttype=fix
pause
Tachyon.Policy.exe -action=import -object=fragments -file=Fragments\*-precondition-*.xml -
fragmenttype=precondition
pause
Tachyon.Policy.exe -action=import -object=rules -file=Rules\*.xml
pause
Tachyon.Policy.exe -action=import -object=policies -file=Policies\*.xml
```

The -fil e

Command line syntax

Commands take the general form:

```
Tachyon.Policy.exe -Server=serverUrl -Object=objectType -Action=action <action-specific-options...>
```

Mandatory parameters

You must specify the following parameters for all actions. Some actions have additional optional parameters which are discussed in the section below on Action-specific options

-Server

The -Server parameter specifies the URL of the Tachyon server's Consumer API endpoint. For example:

```
https://tachyon4.urth.local/Consumer
```

You can omit the -server option if you set an environment variable TACHYON_CONSUMER_URL

-Object

The -Object parameter specifies the policy object type It can be one of the following values:-

- Fragments
- TriggerTemplates
- Rules
- Policies

-Action

The action parameter specifies the desired action you wish to perform. Valid actions are

- **Import** - validate and import definition(s) from one or more XML file(s)
- **Validate** - validate definition(s) from XML file(s)
- **List** - Show a summary list of the policy objects of a given type on the server
- **Delete** - Delete policy object with given ID from the server
- **References** - Show policy objects referencing a given object specified by its ID
- **Export** - generate XML file(s) for the policy object specified by its ID

Action-specific options

Some actions have specific options which are applicable to that action. You specify these options as shown below

-Action=Import or -Action=Validate

The following additional options are required for these actions

-File=<xml source file>

This option defines the name of the source file for the import or validate action. You can specify this option multiple times to process several files. You can also specify wildcards in any part or parts of the file name. For example

-file=Fragments*-check-*.xml

-FragmentType <check|fix|precondition>

If the -Object parameter specified the policy object type as Fragments, this option defines the type of fragment.

-SkipExisting=<true|false>

Specifies whether policy objects already existing on the Tachyon database will be skipped during import or validation. If not specified, the default is false. Specifying SkipExisting=true will mean that any files being validated or imported, that match an existing policy object, will not be processed.

You cannot force the replacement of an existing policy object when specifying -Action=Import. If SkipExisting is false and the policy object already exists, you will receive an error message and the pre-existing policy object will remain unchanged.

-Action=Delete, References or Export

The following additional options are required for these actions

-Id=<id|id1,id2..|*>

Specifies the Id(s) of the object instance. The Id is assigned automatically when the instance is created. To view Ids, use the -list option

You can specify a single Id, a comma-separated list of Ids, or the wildcard symbol (*) to specify all Ids

Examples

List all trigger templates

```
Tachyon.Policy.exe -Server=https://tachyon.acme.local/Consumer -Object=triggertemplates -Action=list
```

Export all fragments

```
Tachyon.Policy.exe -Server=https://tachyon.acme.local/Consumer -Object=fragments -Action=export -Id=*
```

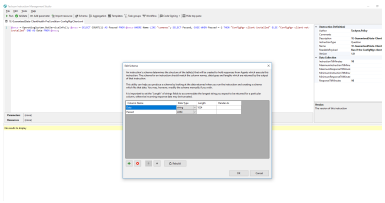
Delete policy with Id=7

```
Tachyon.Policy.exe -Server=https://tachyon.acme.local/Consumer -Object=policies -Action=delete -Id=7
```

Show all references to fragment IDs 88, 89 and 90

```
Tachyon.Policy.exe -Server=https://tachyon.acme.local/Consumer -Object=fragments -Action=references -Id=88,
89,90
```

Source XML files



Fragments

Fragments contain the following elements:

- **Name** - this is the unique name of the fragment, like the Name of an instruction
- **Type** - whether it is a Check, a Fix, or a Precondition - similar to the InstructionType for normal instructions
- **ReadablePayload** - this is friendly text which is displayed in the UI with parameters replaced with their values
- **Payload** - this is the actual SCALE payload that gets executed, just like the Payload of a regular instruction
- **ParameterJson** - these are the parameters for the Fragment. This works exactly like a regular instruction.

These instructions must have two return columns (SchemaJson):

- **Passed** (Boolean) - whether or not the fragment evaluated successfully
- **Data** (String) - additional data only used for displaying to the user

Fragment source XML files are valid files for processing using the TIMS instruction authoring tool.

An example loaded fragment XML file is shown opposite.

> ⚠ The output schema must be as shown. It is assumed that any fragment will return a Data column and a Passed column.

A fragment can be any valid set of SCALE instructions. However, if a fragment attempts to download a resource such as an executable program e.g by using HttpGetFile, the fragment will fail when executed in the context of Guaranteed State. This is because currently when a fragment is imported, the corresponding embedded resource(s) are not uploaded to the background channel.

You can mitigate this by also uploading the fragment as a regular Tachyon instruction. If you do this, it will need to be signed (fragments currently do not require signing). If you upload the fragment this way, then any resources are pushed to the background channel. Subsequently, if the fragment is executed in the context of Guaranteed State, it will successfully retrieve the embedded resource. Be careful when doing this that you manage changes to the embedded resource carefully. If its size and hash change, you will need to ensure that you re-upload the fragment as an instruction and move its release forward to ensure that Tachyon updates internally.

Rules

A sample rule file is shown below. Rule XML files are not valid TIMS files. You will have to currently create them by directly authoring the XML

The policy pack distribution contains a set of initial rule files. You can examine these to determine how to create new rules, based on their structure.

Policies

A sample policy file is shown below. Policy XML files are not valid TIMS files. You will have to currently create them by directly authoring the XML

The policy pack distribution contains a set of initial policy files. You can examine these to determine how to create new policies, based on their structure.

Trigger Templates

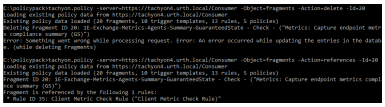
A sample trigger template file is shown below. As for Rules and Policies, you must author these by directly creating the XML.

Trigger templates can take one or more JSON parameters. These are defined differently than the simpler parameters that are associated with rules. An example of such a template is shown below.

The format of the parameter block is identical to parameters in TIMS. Therefore you can, if you wish, create the desired parameters by authoring a TIMS instruction (with no operational SCALE code), define appropriate parameters in TIMS and then copy and paste the UserParameterJson element into your XML

```
<?xml version="1.0" encoding="UTF-8"?>
<TriggerTemplate Name="ServiceStatusChange" Category="Event" Description="When the state of a Windows
service changes" Type="ServiceStatusChange" ReadablePayload="On change of running state of the &amp;quot;%
serviceName%&amp;quot; service">
  <AgentParameters>
    <Parameter Name="ServiceName" Value="%serviceName%" />
  </AgentParameters>
  <UserParameterJson><![CDATA[[{"Name":"ServiceName","Pattern":"%serviceName%","DataType":"string","
ControlType":"freeText","ControlMetadata":null,"Placeholder":"short name of service (e.g. NomadBranch)","
DefaultValue":null,"Validation":{"Regex":null,"MaxLength":"1024","AllowedValues":null,"NumValueRestrictions":
null},"Value":null,"HintText":null,"Source":null}]]]></UserParameterJson>
</TriggerTemplate>
```

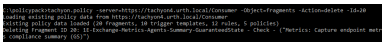
Updating an existing policy object



If you update an existing policy object, you will have to delete all referenced components first. For example, suppose we want to update a fragment which is already associated with a rule and a policy. If we attempt to delete it, we receive an error. Checking the dependencies, we see that the rule with Id 35 references this fragment, so we need to delete it first.

Having deleted the parent rule, we confirm that the fragment no longer has any active references.

We can now delete the fragment.



Now we can import the modified fragment.

At this point the modified fragment is not associated with any rules. Furthermore, the policies (if any) which contained rules associated with the fragment will have had those rules deleted.

Therefore we need to either re-establish the rule, and then associate it with the policies we intend to use it. We can do this manually via the guaranteed state administration UI or we could choose to programmatically import the rule again and then associate it with one or more policies if we wished.

We then need to re-deploy policies from the administration UI for the revised rules/fragments to take effect.

Tachyon Platform zip

The **TachyonPlatform** zip file can be downloaded from the 1E support portal page (https://1eportal.force.com). Extracting the zip will create a folder structure containing the following, where highlighted files are required by Tachyon Setup.

- Licenses.txt
- Tachyon Release Information.html
- **Tachyon.Setup.exe**

- Installers**1ECatalog.msi**
- Installers**SLA.BI.Installer.msi**
- Installers**SLA.Platform.Installer.msi**
- Installers\TachyonCertificateManager.exe
- Installers**TachyonServer.msi**
- Installers\TachyonToolkit.msi

- Installers\Apps\Explorer**Explorer.zip**
- Installers\Apps\Explorer**metadata.json**
- Installers\Apps\GuaranteedState**GuaranteedState.zip**
- Installers\Apps\GuaranteedState**metadata.json**
- Installers\Apps\PatchSuccess**metadata.json**
- Installers\Apps\PatchSuccess**PatchSuccess.zip**
- Installers\Apps\Settings**metadata.json**
- Installers\Apps\Settings**Platform.zip**

- PolicyTool\delete_all.bat
- PolicyTool\Export_All.bat
- PolicyTool\import_all.bat
- PolicyTool\log4net.dll
- PolicyTool\Newtonsoft.Json.dll
- PolicyTool\Tachyon.Policy.exe
- PolicyTool\Tachyon.Policy.exe.config
- PolicyTool\Tachyon.Policy.exe.RoslynCA.json
- PolicyTool\Tachyon.SDK.Consumer.dll
- PolicyTool\Fragments\1E-GuaranteedState-*.xml
- PolicyTool\Policies\Policy-Windows Client Health.xml
- PolicyTool\Rules\Rule-*.xml
- PolicyTool\TriggerTemplates\TriggerTemplate-*.xml