Ex 1 - Tachyon v5.1 Advanced - TIMS Explained

On this page:

TIMS Explained

Tachyon Instruction Management Studio (TIMS) is a graphical user interface for authoring and testing Tachyon instructions and Guaranteed State fragments, which can then be saved so they can be uploaded and used in your Tachyon platform.
The instruction definition XML files that TIMS creates can be uploaded into Tachyon using the Tachyon Explorer so that they can then be used within Tachyon. The Tachyon Product Pack Deployment Tool can also be used to upload objects into Tachyon. The objects for Guaranteed State must be in a product pack that has a specific format.
TIMS makes it easy for you to:

- Run instructions and verify the results that they return on your endpoint device
- Manage resources (such as files) which are associated with the instruction (question or action) you are developing
- Save an instruction, and resources, as an XML file, which can be loaded into Tachyon and made available as a question or action
- Sign an instruction with a different code-signing certificate

TIMS contains a sandboxed copy of the Tachyon Agent, isolated from the Tachyon Agent installed on your computer. TIMS also has its own Agent Historical Data store, which we will work with later in the labs.
Each version of TIMS is bound to the version of its 1E Client and does not have be to the same version as your local Agent, nor the same architecture. This means you can have multiple versions of TIMS installed, so that you can test your instructions against different versions of the 1E Client that you may have in your Tachyon system or are planning to have. However, you can only ever run a single copy of TIMS at any time.

TIMS is only supported on Windows systems. There is no version of TIMS which supports *nix or Android devices.

Configuring TIMS prerequisites

Tachyon requires a code signing certificate to use to sign instructions. This ensures that any instructions written for execution via Tachyon are signed, and no rogue instructions are introduced to the environment. The Code Signing certificate is also associated with the Tachyon license key to ensure that only those with the appropriate certificate can write instructions that can be used in Tachyon.

Enrolling a code signing certificate on the Tachyon server

You do not need a certificate to work within TIMS, however to take an Instruction built in TIMS into a Tachyon environment, you must have a code signing certificate implemented as detailed in this exercise. We have created a code signing certificate for the lab environment.

1ETRNAP

1. Log into **1ETRNAP** as **1ETRN\AppInstaller**
2. Open *SkyTap Shared Drive* on the desktop and navigate to **1E Tachyon - Course Content\Tachyon 5.1 Course Content** download and copy **Tims.v5.1.0.922.zip, SetSCCMSite.ps1** and **1ETRN_Code_Signing_Cert.pfx** to **c:\temp** right click and extract all on the **Tims.v5.1.0.922.zip**
3. From the start menu, type **cert**, and click on **Manage Computer Certificates**
4. In the Certificates console, right-click on **Trusted Publishers.** Select **All Tasks > Import**
5. On the *Welcome,* ensure **Local Machine** is selected by default. Click **Next**
6. On the *File to Import* page, click **Browse**
7. Switch the file type to **All Files (.)**
8. Navigate to **c:\temp** and double-click **1ETRN_Code_Signing_cert.pfx.** Click **Next**
   On the Tachyon server, you only require the .cer file for the certificate. We will use the .pfx for sake of simplicity.
9. On the *Password* page, input **Passw0rd** for the password and click **Next**
10. On the *Certificate Store,* ensure **Trusted Publishers** is populated. Click **Next**
11. On the final page, click **Finish.** Click **OK**
12. In the Certificates console, expand **Trusted Publishers>Certificates** and validate that the code signing cert issued to **SCCM Admin** is present
13. From a command prompt, run the following command

```
iisreset
```

Validate the certificate

14. Double-Click the Code Signing certificate issued in the Trusted Publishers store
15. Click on the **Details** tab and scroll down to the **Thumbprint** value. Make note of the first 4 and last 4 characters of the thumbprint
16. Navigate to **c:\programdata\1E\licensing** and right-click on the **Tachyon.lic** file and open with Notepad
    Use Word Wrap to get a readable view of the file.
17. Do a search for **1ETRN.** Note the **signersha** value associated with the Instruction. Ensure it matches the thumbprint of our code signing certificate
    When a license key is requested from 1E, you must specify what you want your instructions prefix to be and provide the thumbprint for the code signing certificate that will be used to create Instructions. This ensures that only those with the Code Signing certificate will be allowed to create Instructions that will be signed and allowed to be used in Tachyon. This also locks down what the Instruction can be named, in our case the prefix must be **1ETRN**.
18. Close the license file, ensuring no changes were made
19. Close the certificates console

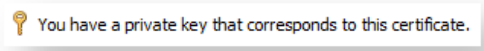
Enrolling the Code Signing certificate on a workstation

TIMS can be run on any computer in your environment. The instructions you develop will use the framework of the machine you are running TIMS on to validate the instruction. As such, we will run TIMS on one of the workstations. Wherever TIMS is run, the same Code Signing certificate must be enrolled in either the User or Computer store, generally in both.

1ETRNW73

20. Log into **1ETRNW73** as **1ETRN\Tachyon_AdminG**
21. Copy **Tims.v5.1.0.922** and **1ETRN_Code_Signing.pfx** from **\\1ETRNAP\temp to c:\tools**
22. From the start menu, type in **mmc** into the search box and launch **mmc.exe**
23. Click **File** and select **Add/Remove Snap-in**
24. From the snap-in wizard, select **Certificates** and click **Add**
25. Select **My user account** and click **Finish click OK**
    If we are running TIMS under system context, the certificate must also reside under the Computer store. In our exercises, we will be running TIMS only in user context.
26. Expand **Certificates (Current User)** and right click on **Personal**
27. Click **All Tasks > Import**
28. On the *Welcome* page, click **Next**
29. On the *File to Import* page, click **Browse**
30. Switch the file type to **All Files (.)**
31. Navigate to **c:\tools** and double-click **1ETRN_Code_Signing_cert.pfx.** Click **Next**
32. On the *Password* page, input **Passw0rd** for the password and click **Next**
33. On the *Certificate Store* click **Next**
34. On the final page, click **Finish.** Click **OK**
35. Double-click the **Code Signing certificate** in the *personal store validate that the code signing cert issued to* ***SCCM Admin*** *is present*

On the *General* page validate that you can see the following below the valid from date



36. On the *Details* page check the thumbprint and validate it is the same as the one on the server and in the license file.

TIMS

Install TIMS on a workstation

In this exercise, we will install TIMS on a workstation, and use that workstation to write instructions.

1ETRNW73

37. Navigate to **c:\tools\Tims.v5.1.0.922** and double-click **TIMS-x86.msi**
    All our workstations are 32 bit in the labs so we will use the x86 version of TIMS. If you were creating instructions on a 64 bit client, you would use the x64 version.
38. On the *Welcome* page, click **Next**
39. On the *License Agreement* page, select **I accept the terms of this license agreement** and click **Next**
40. On the *Destination Folder* page, click **Next**
41. On the *Ready to Install the Program* page, click **Install**
42. Once installed, click **Finish**

Navigating TIMS

Now that we have prereqs in place and TIMS installed, we can begin working with TIMS. In this exercise, we will learn some basics about TIMS and begin exploring the native language, **SCALE (Simple Cross-platform Agent Language for Extensibility),** in the form of Modules and Methods used to write instructions.

1ETRNW73

43. Log in as **1ETRN\Tachyon_AdminG** if not already logged in
44. From the desktop, launch **Tachyon Instrumentation Management Studio**
    Instructions can be written under the user context or the System context. If we wanted to write instructions under the Local System context, we can launch TIMS using PSEXEC to run under the system context. For our labs, we will only use TIMS in the user context.
    TIMS will default to Code Signing certificates. We only have one enrolled, so it will default to that certificate.
45. Review the different buttons in the top pane
46. Click on **Code Signing** and click **Select signing certificate**
47. Click on the **Click here to view certificate properties** link
48. Note that the general tab of the certificate shows **You have a private key that corresponds to this certificate**
    If you do not have a private key associated with the certificate, Instructions written using the certificate will not work in Tachyon.
49. Click **OK** to close the certificate properties. Click **OK** to select the certificate
50. Click on **Code Signing** and check **Always sign**
    This setting ensures that any instruction you attempt to save will be signed using the certificate we just selected.
51. In the top query window, type in the following text

```
SELECT "hello"
```

    Note the query window begins to autofill the text you are typing. This is due to IntelliSense being integrated with TIMS. IntelliSense features are powered by a language service. A language service provides intelligent code completions based on language semantics and an analysis of your source code. If a language service knows possible completions, the IntelliSense suggestions will pop up as you type. If you continue typing characters, the list of members (variables, methods, etc.) is filtered to include only members containing your typed characters. Pressing Tab or Enter will insert the selected member.
52. Click the **Run** button in the ribbon.
    Note that an error box pops up and a syntax error is displayed in the results pane. This is because each line of code must be ended with a semicolon.
53. Click **OK** on the error box
54. At the very bottom, switch from **Results** to **Log** tab
55. Note the various rows of errors. Specifically, a row displaying the following: **Completely unterminated SELECT at line 1,**
56. Return to the **TIMS Query** Add a semicolon to the end of the instruction and click **Run** again
57. Note there are no errors on the **Log** pane
58. Switch to **Results** pane
    Note that the results show a column named hello with single quotes, and a value of hello.
59. Modify the syntax of the query to the following and click **Run**

```
SELECT "hello" AS greeting;
```

60. Note the column name is now set to greeting.
61. Change the query to the following and rerun

```
SELECT "hello" greeting;
```

62. Note that you get the same result as before. The 'AS' is not necessary. If you are familiar with SQL language, you will get the hang of TIMS scripting very quickly.
63. Input the following below the existing line and click **Run**

```
SELECT "goodbye" greeting;
```

64. Note only the second line displays an output. Unlike SQL, both rows will not display an output, only the last. Switch the rows around to see only 'hello' will show. This is very important to remember when you begin to write complex instructions.
65. Now that you have done a simple SELECT query in which the data returned is hard-coded, let's run a query using SCALE. Clear the existing code from TIMS. Input the following query into the window. Type it out exactly as shown below, do not rely on IntelliSense to autofill it

```
Network.getconnections();
```

66. Note the error returned. **Unsupported method 'getconnections'.**
67. Change the syntax to the following

```
Network.GetConnections();
```

68. Note the results that are returned. The scripting language is case sensitive, so you must input very exact syntax. IntelliSense will ensure that these syntax errors are autocorrected.
69. Clear the query window and run the following queries, one line at a time, and review the results

```
Software.GetInstallations();
```

```
Device.GetSummary();
```

```
FileSystem.GetFilesInFolder(Folder: "c:\\program files\\1E\\client\\");
```

Notice the \\ in the file path. In strings use the sequences backslash-double-quote (\") and backslash-backslash (\\) to escape literal double-quotes (") and a literal backslash (\) respectively.

```
OperatingSystem.GetServiceInfo();
```

```
OperatingSystem.GetInstalledUpdates();
```

These queries give you a general sense of how the Methods and corresponding Modules operate. You can review all the Methods and Modules and use them in different ways to get data back. Subsequently, you can do more than just query machines, you can also take action on machines. We will cover that later as we begin to write a complex Instruction using different Methods and Modules as well as functions.

Creating a Basic Instruction

Create Basic Instruction

70. Clear the TIMS query window and run the following

```
Network.GetConnections();
```

71. Let's add a comment to our instruction. Above the first line Type in

```
//All the Active Connections on this device
```

/* This is a comment until the block is closed */ and // This is a comment until the end of the line. Comments are useful to annotate the instruction but serve no useful purpose on the device. Since they are ignored by the device when running an instruction omitting them could save network traffic in large environments.
72. Click on the **Schema** button on the top ribbon. On the popup, click **Yes**.
73. In the Schema editor, click the different boxes. Here you can define the format of the responses that come back from the devices.
The **GetConnections** method has a predefined schema. We can define that value in the schema itself manually. The **render as** property allows for 3 rendering schemas currently, **perforated, IPAddress,** or **currency.** This allows for the results to be shown rendered as one of these data types.
74. Click on **ProcessName** and using the  button, move it to the top.
75. Click **OK** to save the schema
76. Click on **Workflow**. Note the different options available for Instruction Workflow in the dropdown. Click **Cancel**
The workflow is defined as a global setting, but we can override that by explicitly defining a workflow per instruction. Certain Instructions might be low risk, and thus not require approval or 2FA.
77. Click **Task groups**
Tasks allow us to add instructions to specific task groups so they can be executed directly from the **Tasks** workspace in the Tachyon explorer console. This allows them to be run on a schedule.
78. Click **Cancel** to exit out of Task Groups
79. Click on the **Aggregation** button on the ribbon
Here, we can group by a specific value that is returned, as well as aggregate based on the available operations. We will work with aggregation later in the labs.
80. Click **Cancel**
81. Click **Validate** in the top ribbon. Note the errors displayed. Click **Close**
A readable payload must be defined, and a description for the Instruction provided so it can be searched against in the Tachyon explorer console. A payload is essentially what the Instruction will present to the operator via the explorer console. A task group is not required; hence it is only a warning.
82. In the **Instruction Definition** panel at the top right, note the different properties
83. In the **Description** field, input **Active Network Connections** and click **Enter**
You can click on the ellipses in the field to open a dialog box as well.

84. For the **Name** field**,** change the value from the default to **1ETRN-ActiveNetworkConnections**
85. For the **ReadablePayload** field, click on the ellipses at the right and input **Active Network Connections.** Click **OK**
86. Review the other properties in the Instruction Definition pane**.** These are all configurable parameters, but if left alone they will default to the values of the installation.
87. Click **Validate** in the top ribbon. Note only a task groups warning is shown. Click **Close**
88. From the menu bar, click **File > Save As.** Click **OK** on the validation page
89. Save the Instruction under **c:\tools** as **1ETRN-ActiveNetworkConnections.xml**
90. Note in the top pane a message is displayed stating that the instruction definition was signed by XXX. This is because we chose to **Always Sign** earlier. Leave TIMS open with the instruction still defined in the top pane

Validating the Instruction

91. In windows explorer, navigate to **c:\tools**. Note an XML file named **1ETRN-ActiveNetworkConnections** has been created here
92. Right-click the file and open with Internet Explorer
93. Note the format and content of the instruction file
    We have reviewed instruction XML files in the Tachyon course, this format should look familiar.
94. Open the ConfigMgr Content Location shortcut from the desktop. Copy **c:\tools\1ETRN-ActiveNetworkConnections.xml** to the ConfigMgr Content folder
    We need to import our instructions with an account that has Instruction Set permissions in Tachyon.

Uploading the instruction in Tachyon

In this task, we will add the newly created Instruction to Tachyon and ask the question to our clients.

1ETRNAP

95. Log into **1ETRNAP** as **1ETRN\AppInstaller**
96. Open the *ConfigMgr Content Source location* shortcut on the *desktop*. Copy **1ETRN-ActiveNetworkConnections.xml** to **c:\temp**
97. Open *Google Chrome* navigate to **HTTPS://Tachyon.1ETRN.local/Tachyon** and launch the *Settings* application
98. Navigate to *Instructions – Instruction Sets*
99. Click the Upload button at the top and browse to **c:\temp** and select **1ETRN-ActiveNetworkConnections.xml**
100. Click **Open.** Click on **Recent Uploads** Ensure the Instruction is uploaded successfully and there are no errors
101. Click back on **Instruction Sets**. Select the Unassigned Instruction Set. Find the new instruction (you may need to show 48 items and resort the list).
102. Select the *Active Network Connections* instruction and click on the **Add New Set** button on the right pane.
103. In the *Add new instruction set* dialog box type **1ETRN** in the *Name* field. Ensure that **Include 1 Selected Instructions** is checked. Click **Add**.
104. Click on the **1ETRN** Instruction Set in the left pane to validate the instruction has been moved there
     We are now ready to execute this Instruction on our client machines. All instructions must reside within an Instruction Set before they can be executed via the Explorer Application.

Running the newly created Instruction

1ETRNW71

105. Log into **1ETRNW71** as **1ETRN\Tachyon_Admin1**
106. Open *Google Chrome* Launch the *Tachyon Portal* from https://Tachyon.1etrn.local/Tachyon
107. Launch the *Explorer* application
108. From the top right, click the **All instructions** button
109. Note a **1ETRN** Instruction set is now present. Expand the Instruction Set
110. Click on the instruction and click **Ask this question**
     The data returned should display **ProcessName** as the second column since we changed the schema to move it to the top. If we left the schema default, the results would show in the same order they are shown in TIMS, except for the device name column (this is returned as the first column by default from the device that gave the responses).
111. Scroll down to see the results returned from the different machines
112. Click the **Back to Top** link at the bottom right
113. Click on the **Filter Results** button to expand the filter options
114. In the **Device Name** box, input **1ETRNW73** for the local machine. Click **Search**
115. Review the results for the local machine
116. Return to TIMS. With the **Network.GetConnections();** query still defined, click the **Run** button
117. Note the connections. They should match what Tachyon returns via the Instruction although the column order will be slightly different.

Lab Summary

In this lab, we learned what TIMS is, and the native language, SCALE it uses. We installed and configured the prerequisites required to ensure Instructions created by TIMS will be accepted by Tachyon, and we then installed TIMS. We then evaluated some Methods and Modules to understand how the native language works, using some basic examples. Lastly, we created an Instruction and ran it via Tachyon to review the results returned.

Next Page
Ex 2 - Tachyon v5.1 Advanced - TIMS Functions