Set up in instruction to be executed on a schedule

Introduction

This page describes how Tachyon instructions can be set up to execute on the schedule.

You should also read Issue an instruction, track its progress and retrieve responses page, which covers tracking instruction's progress, retrieving responses and the like, because here we will only focus on the scheduling aspect.

The C# examples assume you're using Tachyon Consumer SDK and you already have a correctly instantiated instance of Tachyon connector class in an object called connector.

All SDK methods return the same class called ApiCallResponse. Inside the object of ApiCallResponse you'll find a property called ReceivedObject. That object is the actual data received from the API. In the following examples this detail is left out, stating that the returned object contains the data. For example, when we say that XYZ object contains certain data, this means the ReceivedObject contains that data, since that's always true.

On this page

What is a "scheduled instruction"

A "scheduled instruction" is a regular Tachyon instruction definition that has been configured to execute periodicaly on a specific schedule."Scope" property to the schedule creating payload

When a schedule is set up, you define the instruction as you would normally (see Issue an instruction, track its progress and retrieve responses), with parameters, scope etc. and add information about the schedule.

When the time comes to execute the schedule, Tachyon will issue a new copy of the instruction. This will be a carbon copy of the instruction that was scheduled - all fields will be copied, including auxiliary ones like Comments and ConsumerCustomData. This means that each copy of the instruction issued on a schedule will have the same parameters, the same scope or target devices etc.

Creating a schedule

To create a schedule you have to provide the template for the instruction that will be issued when the schedule is triggered and information about the schedule itself.

It is worth noting that creating a schedule will not result in an instruction being issued immediatelly. This is because instructions are issued only when a schedule is triggered.

A schedule should be created with care as it is possible to create one that will flood the system with requests. It is not advised to create a schedule where a new instruction is issued before the previous' ones instruction ttl expires.

Because Tachyon's scheduling mechanism is relatively simple, if you require features like variation in instruction parameters or targeting between instructions triggered on a schedule, you might want to look at implementing scheduling yourself and issuing regular instructions instead.

Core of the scheduled instruction

The core of the scheduled instruction is much the same as a regular instruction, as seen in Issue an instruction, track its progress and retrieve responses.

Each instruction issued on the schedule will be a copy of the instruction provided when a schedule is created.

Scheduling properties

Schedule properties with the exception of ScheduleEnabled are based on Microsoft SQL Server system schedules as described in details here: https://docs.microsoft.com/en-us/sql/relational-databases/system-tables/dbo-sysschedules-transact-sql?view=sql-server-ver15

Schedule property name	Microsoft SQL column name	Description
ScheduleEnabled	enabled	This field defined if a schedule is enabled or not. Schedules that are not enabled will not be executed. If a schedule is enabled after some of the occurrences should have happened, the schedule will resume from the next scheduled occurence and will not "catch up".

<table>
<tr><td>ScheduleFreqType</td><td>freq_type</td><td>How frequently a job runs for this schedule.
Value of 1 means only there will be only one activation of this schedule so only one instruction will be issued
Value of 4 means daily activation
Value of 8 means weekly activation
Value of 16 means monthly activation
Value of 32 means monthly activation relative to ScheduleFreqInterval</td></tr>
<tr><td>ScheduleFreqInterval</td><td>freq_interval</td><td>Meaning of the value of this fields depends on the value of ScheduleFreqType.
If ScheduleFreqType is set to 1 (one activation of the schedule), this field is not used and should have the value of 0
If ScheduleFreqType is set to 4, this field will have the number of days between activations of the schedule. For example, to make a schedule activate every day, set this to 1. To do it every other day, set it to 2.
If ScheduleFreqType is set to 8, this field becomes a bit flag that contains information on which days of the week the schedule should be activated. The values are as follows:
<table><tr><th>Day</th><th>Value</th></tr><tr><td>Sunday</td><td>1</td></tr><tr><td>Monday</td><td>2</td></tr><tr><td>Tuesday</td><td>4</td></tr><tr><td>Wednesday</td><td>8</td></tr><tr><td>Thursday</td><td>16</td></tr><tr><td>Friday</td><td>32</td></tr><tr><td>Saturday</td><td>64</td></tr></table>For example, if you wanted to set the schedule up to activate every day of the week the value of ScheduleFreqInterval would be 127. If you only wanted the schedule to activate on Mondays and Fridays it would be 34.
If ScheduleFreqType is set to 16, the value of this field will be the number of the day of the month on which the schedule will be activated.
If ScheduleFreqType is set to 32, this propert will have one of following values:
<table><tr><th>Description</th><th>Value</th></tr><tr><td>Sunday</td><td>1</td></tr><tr><td>Monday</td><td>2</td></tr><tr><td>Tuesday</td><td>3</td></tr><tr><td>Wednesday</td><td>4</td></tr><tr><td>Thursday</td><td>5</td></tr><tr><td>Friday</td><td>6</td></tr><tr><td>Saturday</td><td>7</td></tr><tr><td>Day</td><td>8</td></tr><tr><td>Weekday</td><td>9</td></tr><tr><td>Weekend day</td><td>10</td></tr></table></td></tr>
<tr><td>ScheduleFreqSubdayType</td><td>freq_subday_type</td><td>Units for the ScheduleFreqSubdayInterval.
Value of 1 means activate at the specified time
Value of 2 means the unit is Seconds
Value of 4 means the unit is Minutes
Value of 8 means the unit is Hours</td></tr>
<tr><td>ScheduleFreqSubdayInterval</td><td>freq_subday_interval</td><td>Number of ScheduleFreqSubdayType to occur between each activation of the schedule.</td></tr>
</table>

<table>
<tr><td>ScheduleFreqRelativeInterval</td><td>freq_relative_interval</td><td>If ScheduleFreqInterval is set to 32 this property can have one of following values:<table><tr><th>Description</th><th>Value</th></tr><tr><td>ScheduleFreqRelativeInterval is not used</td><td>0</td></tr><tr><td>First</td><td>1</td></tr><tr><td>Second</td><td>2</td></tr><tr><td>Third</td><td>4</td></tr><tr><td>Fourth</td><td>8</td></tr><tr><td>Last</td><td>16</td></tr></table>If ScheduleFreqInterval is set to any other value this field is ignored.</td></tr>
<tr><td>ScheduleFreqRecurrenceFactor</td><td>freq_recurrence_factor</td><td>Number of weeks or months between the activation of the schedule. ScheduleFreqRecurrenceFactor is used only if ScheduleFreqType is set to 8, 16, or 32. If this column contains 0, ScheduleFreqRecurrenceFactor will not be used.</td></tr>
<tr><td>ScheduleActiveStartDate</td><td>active_start_date</td><td>Date on which the schedule should begin formatted as YYYYMMDD</td></tr>
<tr><td>ScheduleActiveEndDate</td><td>active_end_date</td><td>Date on which the schedule should finish formatted as YYYYMMDD</td></tr>
<tr><td>ScheduleActiveStartTime</td><td>active_start_time</td><td>Time between ScheduleActiveStartDate and ScheduleActiveEndDate which will be used as starting time for the schedule exexution on a given day. Time is formatted HHMMSS, using a 24-hour clock.</td></tr>
<tr><td>ScheduleActiveEndTime</td><td>active_end_time</td><td>Time between ScheduleActiveStartDate and ScheduleActiveEndDate which will be used as stopping time for the schedule exexution on a given day. Time is formatted HHMMSS, using a 24-hour clock.</td></tr>
</table>

A schedule can be created disabled by setting the ScheduleEnabled property to false. This will mean that it will not be triggered and no instructions will be issued until the schedule is enabled. If the schedule is enabled at a point in time where it would have already missed some activations, it will simply resume from the next scheduled activation. For example, if on Monday morning you create a disabled schedule that is meant to trigger every day at mid day and then go ahead an enabled it on Wednesday at 5pm, the next triggered activation will be on Thursday at mid day, and the schedule would have "missed" activations on Monday, Tuesday and Wednesday because it was disabled. This means that an instruction will be issued at mid day Thursday, but there will be no instructions for Monday, Tuesday or Wednesday.

Managing schedules

In this section we will learn how to examine what schedules are available, how to create, modify and cancel them and how to see what instructions will be issued by a schedule.

Retrieving scheduled instructions

To retrieve all scheduled instructions (but not instructions issued by schedules) you should use the https://my.tachyon.server/Consumer/ScheduledInstructions endpoint.

Direct Consumer API call	**C# code using Consumer SDK library**
Making a GET request to https://my.tachyon.server/Consumer/ScheduledInstructions will return all scheduled instructions:	

Return payload


```
[
    {
        "Id": 1,
        "Name": "1E-Explorer-TachyonAgent-Echo",
        "Cmd": "SendAll",
        "KeepRaw": true,
        "Scope": null,
        "ReadableScope": null,
        "InstructionTtlMinutes": 60,
        "ResponseTtlMinutes": 120,
        "CreatedTimestampUtc": "2019-12-06T10:53:31.903Z",
        "Export": false,
        "ExportLocation": null,
        "ParentInstructionId": null,
```

```
        "InstructionDefinitionId": 23,
        "CreatedBy": "SomeDomain\\Jane.Doe",
        "ResultsFilter": null,
        "PreviousResultsFilter": null,
        "Devices": null,
        "ConsumerId": 1,
        "ConsumerName": "Explorer",
        "ConsumerCustomData": null,
        "ParameterJson": "[{\"Name\":\"msg\",\"Pattern\":\"%msg%\",\"DataType\":null,\"
ControlType\":null,\"ControlMetadata\":null,\"Placeholder\":null,\"DefaultValue\":null,\"
Validation\":null,\"Value\":\"Machine reporting\",\"HintText\":null,\"Source\":null}]",
        "OffloadResponses": false,
        "RequestedFor": null,
        "Comments": "Reporting machine echo",
        "ActionedBy": "Ollanius Pius",
        "ActionReason": null,
        "Status": 7,
        "StatusTimestampUtc": "2019-12-10T13:24:35.917Z",
        "Workflow": "{\"StateMachine\":\"Scheduled Instruction State with Authentication
and Approval\",\"Authenticating\":\"Scheduled Instruction Authentication\",\"Approval\":
\"Scheduled Instruction Approval\",\"InProgress\":\"Scheduled Instruction InProgress\",\"
Cancelling\":\"Scheduled Instruction Cancelling\"}",
        "WorkflowState": 8,
        "ApprovalOffloaded": false,
        "ReadablePayload": "Echo message %msg%.",
        "ScheduleEnabled": true,
        "ScheduleFreqType": 4,
        "ScheduleFreqInterval": 1,
        "ScheduleFreqSubdayType": 8,
        "ScheduleFreqSubdayInterval": 12,
        "ScheduleFreqRelativeInterval": 0,
        "ScheduleFreqRecurrenceFactor": 0,
        "ScheduleActiveStartDate": "2019-12-06T00:00:00Z",
        "ScheduleActiveEndDate": "2019-12-09T00:00:00Z",
        "ScheduleActiveStartTime": 120000,
        "ScheduleActiveEndTime": 120000,
        "ScheduleLastExecuted": null,
        "ScheduleNextExecution": null,
        "ScheduleReadableFrequency": "Occurs every 1 day(s) every 12 hours starting at
12:00:00 ending at 12:00:00 starting on 06/12/2019 ending on 09/12/2019"
    },
    {
        "Id": 2,
        "Name": "1E-Explorer-TachyonAgent-Echo",
        "Cmd": "SendAll",
        "KeepRaw": true,
        "Scope": null,
        "ReadableScope": null,
        "InstructionTtlMinutes": 60,
        "ResponseTtlMinutes": 120,
        "CreatedTimestampUtc": "2019-12-10T13:04:06.6Z",
        "Export": false,
        "ExportLocation": null,
        "ParentInstructionId": null,
        "InstructionDefinitionId": 23,
        "CreatedBy": "SomeDomain\\Jane.Doe",
        "ResultsFilter": null,
        "PreviousResultsFilter": null,
        "Devices": null,
        "ConsumerId": 1,
        "ConsumerName": "Explorer",
        "ConsumerCustomData": null,
        "ParameterJson": "[{\"Name\":\"msg\",\"Pattern\":\"%msg%\",\"DataType\":\"
string\",\"ControlType\":\"freeText\",\"ControlMetadata\":null,\"Placeholder\":\"message
to echo\",\"DefaultValue\":null,\"Validation\":{\"Regex\":null,\"MaxLength\":\"200\",\"
AllowedValues\":null,\"NumValueRestrictions\":null},\"Value\":\"echo echo echo\",\"
HintText\":null,\"Source\":null}]",
        "OffloadResponses": false,
        "RequestedFor": null,
        "Comments": null,
        "ActionedBy": "Ollanius Pius",
```


Use ScheduledInstructions object inside the Tachyon connector instance.

Retrieving all scheduled instructions

```
schedules =
connector.
ScheduledInst
ructions.
GetAll();
```

"schedules" object will contain the same data you can see in the JSON response on the left.

```
        "ActionReason": null,
        "Status": 7,
        "StatusTimestampUtc": "2019-12-12T13:27:57.5Z",
        "Workflow": "{\"StateMachine\":\"Scheduled Instruction State with Authentication
and Approval\",\"Authenticating\":\"Scheduled Instruction Authentication\",\"Approval\":
\"Scheduled Instruction Approval\",\"InProgress\":\"Scheduled Instruction InProgress\",\"
Cancelling\":\"Scheduled Instruction Cancelling\"}",
        "WorkflowState": 8,
        "ApprovalOffloaded": false,
        "ReadablePayload": "Echo message %msg%.",
        "ScheduleEnabled": true,
        "ScheduleFreqType": 1,
        "ScheduleFreqInterval": 0,
        "ScheduleFreqSubdayType": 0,
        "ScheduleFreqSubdayInterval": 0,
        "ScheduleFreqRelativeInterval": 0,
        "ScheduleFreqRecurrenceFactor": 0,
        "ScheduleActiveStartDate": "2019-12-10T00:00:00Z",
        "ScheduleActiveEndDate": "2019-12-11T00:00:00Z",
        "ScheduleActiveStartTime": 142000,
        "ScheduleActiveEndTime": 140200,
        "ScheduleLastExecuted": "2019-12-10T14:20:00Z",
        "ScheduleNextExecution": null,
        "ScheduleReadableFrequency": "Occurs once on 10/12/2019 at 14:20:00"
    },
    {
        "Id": 3,
        "Name": "1E-Explorer-TachyonAgent-Echo",
        "Cmd": "SendAll",
        "KeepRaw": true,
        "Scope": null,
        "ReadableScope": null,
        "InstructionTtlMinutes": 60,
        "ResponseTtlMinutes": 120,
        "CreatedTimestampUtc": "2019-12-10T13:06:47.78Z",
        "Export": false,
        "ExportLocation": null,
        "ParentInstructionId": null,
        "InstructionDefinitionId": 23,
        "CreatedBy": "SomeDomain\\Jane.Doe",
        "ResultsFilter": null,
        "PreviousResultsFilter": null,
        "Devices": null,
        "ConsumerId": 1,
        "ConsumerName": "Explorer",
        "ConsumerCustomData": null,
        "ParameterJson": "[{\"Name\":\"msg\",\"Pattern\":\"%msg%\",\"DataType\":\"
string\",\"ControlType\":\"freeText\",\"ControlMetadata\":null,\"Placeholder\":\"message
to echo\",\"DefaultValue\":null,\"Validation\":{\"Regex\":null,\"MaxLength\":\"200\",\"
AllowedValues\":null,\"NumValueRestrictions\":null},\"Value\":\"echo echo echo\",\"
HintText\":null,\"Source\":null}]",
        "OffloadResponses": false,
        "RequestedFor": null,
        "Comments": null,
        "ActionedBy": "Ollanius Pius",
        "ActionReason": null,
        "Status": 7,
        "StatusTimestampUtc": "2019-12-12T13:27:59.543Z",
        "Workflow": "{\"StateMachine\":\"Scheduled Instruction State with Authentication
and Approval\",\"Authenticating\":\"Scheduled Instruction Authentication\",\"Approval\":
\"Scheduled Instruction Approval\",\"InProgress\":\"Scheduled Instruction InProgress\",\"
Cancelling\":\"Scheduled Instruction Cancelling\"}",
        "WorkflowState": 8,
        "ApprovalOffloaded": false,
        "ReadablePayload": "Echo message %msg%.",
        "ScheduleEnabled": true,
        "ScheduleFreqType": 1,
        "ScheduleFreqInterval": 0,
        "ScheduleFreqSubdayType": 0,
        "ScheduleFreqSubdayInterval": 0,
        "ScheduleFreqRelativeInterval": 0,
```

```
        "ScheduleFreqRecurrenceFactor": 0,
        "ScheduleActiveStartDate": "2019-12-10T00:00:00Z",
        "ScheduleActiveEndDate": "2019-12-11T00:00:00Z",
        "ScheduleActiveStartTime": 142000,
        "ScheduleActiveEndTime": 140200,
        "ScheduleLastExecuted": "2019-12-10T14:20:00Z",
        "ScheduleNextExecution": null,
        "ScheduleReadableFrequency": "Occurs once on 10/12/2019 at 14:20:00"
    }
]
```

You can also retrieve a single scheduled instruction by using its Id. You cannot use the name because there might be multiple scheduled instructions that use the same instruction definition.

It is important to remember that this is the Id of the schedule, not the instruction definition on which scheduled instructions are based upon. This example shows how you can locate all schedules based on a specific instruction definition.

Direct Consumer API call	**C# code using Consumer SDK library**

Making a GET request to https://my.tachyon.server/Consumer/ScheduledInstructions/1 will yield following response:

Return payload


```
{
    "Id": 1,
    "Name": "1E-Explorer-TachyonAgent-Echo",
    "Cmd": "SendAll",
    "KeepRaw": true,
    "Scope": null,
    "ReadableScope": null,
    "InstructionTtlMinutes": 60,
    "ResponseTtlMinutes": 120,
    "CreatedTimestampUtc": "2019-12-06T10:53:31.903Z",
    "Export": false,
    "ExportLocation": null,
    "ParentInstructionId": null,
    "InstructionDefinitionId": 23,
    "CreatedBy": "SomeDomain\\Jane.Doe",
    "ResultsFilter": null,
    "PreviousResultsFilter": null,
    "Devices": null,
    "ConsumerId": 1,
    "ConsumerName": "Explorer",
    "ConsumerCustomData": null,
    "ParameterJson": "[{\"Name\":\"msg\",\"Pattern\":\"%msg%\",\"DataType\":null,\"
ControlType\":null,\"ControlMetadata\":null,\"Placeholder\":null,\"DefaultValue\":
null,\"Validation\":null,\"Value\":\"Machine reporting\",\"HintText\":null,\"Source\":
null}]",
    "OffloadResponses": false,
    "RequestedFor": null,
    "Comments": "Reporting machine echo",
    "ActionedBy": "Ollanius Pius",
    "ActionReason": null,
    "Status": 7,
    "StatusTimestampUtc": "2019-12-10T13:24:35.917Z",
    "Workflow": "{\"StateMachine\":\"Scheduled Instruction State with Authentication
and Approval\",\"Authenticating\":\"Scheduled Instruction Authentication\",\"
Approval\":\"Scheduled Instruction Approval\",\"InProgress\":\"Scheduled Instruction
InProgress\",\"Cancelling\":\"Scheduled Instruction Cancelling\"}",
    "WorkflowState": 8,
    "ApprovalOffloaded": false,
    "ReadablePayload": "Echo message %msg%.",
    "ScheduleEnabled": true,
    "ScheduleFreqType": 4,
    "ScheduleFreqInterval": 1,
    "ScheduleFreqSubdayType": 8,
    "ScheduleFreqSubdayInterval": 12,
    "ScheduleFreqRelativeInterval": 0,
    "ScheduleFreqRecurrenceFactor": 0,
    "ScheduleActiveStartDate": "2019-12-06T00:00:00Z",
    "ScheduleActiveEndDate": "2019-12-09T00:00:00Z",
    "ScheduleActiveStartTime": 120000,
    "ScheduleActiveEndTime": 120000,
    "ScheduleLastExecuted": null,
    "ScheduleNextExecution": null,
    "ScheduleReadableFrequency": "Occurs every 1 day(s) every 12 hours starting at 12:
00:00 ending at 12:00:00 starting on 06/12/2019 ending on 09/12/2019"
}
```


Use ScheduledInstructions object inside the Tachyon connector instance.

Retrieving a single scheduled instruction

```
schedule =
connector.
ScheduledInstruc
tions.
GetScheduledInst
ruction(1);
```

"schedule" object will contain the same data you can see in the JSON response on the left.

Retrieving past and/or planned instructions

You can search for schedule activations that match certain parameters that have occurred or will occur between certain points in time.

Before we dwell into examples of how this can be used, let's look at how the API call itelf will look like so that in subsequent examples we can focus on crafting just the search parameters payload to handle specific use-case.

For now, let's locate all scheduled activations for a Consumer whose Id is 7 that will occur in the next week:

Direct Consumer API call

Making a POST request to https://my.tachyon.server/Consumer/ScheduledInstructions/Search

Request payload

```
{
  "Filter": {
    "Attribute": "ConsumerId",
    "Operator": "==",
    "Value": "7"
  },
  "StartDate": "2018-02-14T15:10:22.935Z",
  "EndDate": "2020-02-21T15:10:22.936Z",
  "PageSize": 100
}
```

will yield following response:

Return payload

```
{
    "TotalCount": 2,
    "Items": [
        {
            "Id": 9,
            "Name": "1E-Explorer-TachyonCore-
NetworkAdapterConfigurationDetails",
            "CreatedTimestampUtc": "2020-02-01T09:45:11.377Z",
            "CreatedBy": "SomeDomain\\Jago.Sevatarion",
            "ConsumerId": 7,
            "ConsumerName": "MyCustomConsumer",
            "ConsumerCustomData": null,
            "ParameterJson": null,
            "Comments": null,
            "ScheduleNextExecution": "2020-02-15T09:50:00.047Z",
            "ScheduleReadableFrequency": "Occurs every 1 weeks(s) on Monday,
Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday at 09:50:00 starting
on 01/02/2020 ending on 16/02/2020",
            "ReadablePayload": "How are network adapters configured?",
            "InstructionId": null
        },
        {
            "Id": 9,
            "Name": "1E-Explorer-TachyonCore-
NetworkAdapterConfigurationDetails",
            "CreatedTimestampUtc": "2020-02-01T09:45:11.377Z",
            "CreatedBy": "SomeDomain\\Jago.Sevatarion",
            "ConsumerId": 7,
            "ConsumerName": "MyCustomConsumer",
            "ConsumerCustomData": null,
            "ParameterJson": null,
            "Comments": null,
            "ScheduleNextExecution": "2020-02-16T09:50:00Z",
            "ScheduleReadableFrequency": "Occurs every 1 weeks(s) on Monday,
Tuesday, Wednesday, Thursday, Friday, Saturday, Sunday at 09:50:00 starting
on 01/02/2020 ending on 16/02/2020",
            "ReadablePayload": "How are network adapters configured?",
            "InstructionId": null
        }
    ]
}
```

C# code using Consumer SDK library

Use ScheduledInstructions object inside the Tachyon connector instance.

Scheduled activations in the next 7 days

```
var payload = new
ScheduledSearch
{
    PageSize = 100,
    Filter = new
ExpressionObject
    {
        Attribute =
"ConsumerId",
        Operator = "==",
        Value = "7"
    },
    StartDate = DateTime.
UtcNow,
    EndDate = DateTime.
UtcNow.AddDays(7)
};
activations = connector.
ScheduledInstructions.
Search(payload);
```

"activations" object will contain the same data you can see in the JSON response on the left.

Please take note that while the JSON example has the date-time hard-coded, so it will needs adjusting when run, the C# example takes advantage of the DateTime.UtcNow to always be correct.

Entries returned in the example above have null in "InstructionId" property. This is because we've searched for entries in the future. Also, "CreatedTimestampUtc" has the timestamp of when the schedule was created, since the instruction hasn't been issued yet. If our search returned schedule activations that have already happened, the "InstructionId" would have, as name suggests, the Id of the Instruction that was issued by the schedule activation. In this case, "CreatedTimestampUtc" would have the time stamp of when the instruction was created, and not the schedule.

Let us examine the fields in the payload that we send to the API:

Property	Description
StartDate	Date time defining the start of the search window. If this is not provided the system will assume the "beginning of time" in the form of C# DateTime.MinValue, which according to Microsoft Documentation "is equivalent to 00:00:00.0000000 UTC, January 1, 0001, in the Gregorian calendar."
EndDate	EndDate must be provided if PageSize is not provided. If PageSize is provided then EndDate is optional. This property defines the end of the search window. If it is not provided the system will return enough entries to fill the PageSize provided. If it is provided the system will provide entries that fit between StartDate and EndDate only, even if there are more entries after the EndDate that would fit on the page.
PageSize	PageSize must be provided if EndDate is not provided. If EndDate is provided then PageSize is optional. This property defines how many entries should be returned. If it is not provided, the system will return all entries between StartDate and EndDate. If it is provided it will only return enough entries after the StartDate (but before EndDate, if one is provided) to fill the PageSize.
Filter	This is a filter expression that describes what schedules the system should search for. The API call will then return all activations for those schedules that are between StartDate and EndDate. This is a standard Tachyon expression and you can find more information about them on Using scope and filter expressions page.

Now we will go through a few use-cases. These will focus on the Filter property, since that if what defines which schedules will be taken into account when past activations are gathered and future activations are extrapolated.

Finding all schedule activations for a specific instruction definition

You can use either the Id or Name of the instruction definition to locate all schedules that will use that definition. If I wanted to look for all activations of a schedule that will issue "1E-Explorer-TachyonAgent-AgentDiagnostics" instruciton, which in my installation has the Id of 18, I would create one of these two filters:

Filter by instruction definition Id

```
{
    "Attribute": "DefinitionId",
    "Operator": "==",
    "Value": "18"
}
```

or

Filter by instruction definition Name

```
{
    "Attribute": "DefinitionName",
    "Operator": "==",
    "Value": 1E-Explorer-TachyonAgent-AgentDiagnostics
}
```

Finding all schedule activations for a specific scheduled instruction

When you create a scheduled instruction (see below), you receive its Id. You can then use that Id to track instructions issued by the schedule.

Find all instructions issued for a specific schedule

```
{
    "Attribute": "ScheduledInstructionId",
    "Operator": "==",
    "Value": "1"
}
```

Finding all schedule activations that contain a marker

You can use ConsumerCustomData property in the payload sent to the API when you create a scheduled instruction to create a "marker", which you can then use to find scheduled instructions. This field is not examined or used by Tachyon, but it will be copied from the Schedule into each instance of an instruction issued by said Schedule.

Find all instructions that bear a specific marker

```
{
    "Attribute": "ConsumerCustomData",
    "Operator": "==",
    "Value": "My special instruction marker"
}
```

Complex example

Lastly, let's look at the entire payload (so not just the filter) for one last use case. Let us assume that various consumers schedule instructions and when those instructions are issued they are given a marker which defines what Organizational Unit they are issued by. Now let's assume we would like to find all scheduled activations of instruction definition "1E-Explorer-TachyonCore-DeviceDrivers" in May 2020 for consumer "MyCustonConsumer", which has the Id of 7 that had the marker value of "OU:Finance".

Request payload

```
{
    "Filter": {
        "Operator": "AND",
        "Operands": [
            {
                "Attribute": "ConsumerCustomData",
                "Operator": "==",
                "Value": "OU:Finance"
            },
            {
                "Attribute": "ConsumerId",
                "Operator": "==",
                "Value": "7"
            },
            {
                "Attribute": "DefinitionName",
                "Operator": "==",
                "Value": "1E-Explorer-TachyonCore-DeviceDrivers"
            }
        ]
    },
    "StartDate": "2020-05-01T00:00:00.001Z",
    "EndDate": "2020-05-31T23:59:59.999Z"
}
```

Lifecycle of a schedule

Lifecycle of a schedule is similar to that of an instruction. By default, Tachyon is configured to require authentication and approval for each schedule. Schedule will be "live" only once creating user has authenticated themselves and the schedule has been approved, providing is has been created with the ScheduleEnabled flag set to true.

A schedule will remain live between its starting and finishing dates, so between "ScheduleActiveStartDate" and "ScheduleActiveEndDate". How the schedule is triggered/activated between those dates depends on its settings. Once the schedule finishes, its Status will be set to Expired (7).

If the user creating a schedule fails to authenticate is good time, the schedule will fail. If the user authenticates, the schedule then needs to be approved and a schedule will remain in "InApproval" (1) state until it is either rejected or approved, so potentially can be stuck for a long time.

The requirement to authenticate the user creating a schedule isn't ideal if these schedules are created programmaticaly.

To alleviate that, you can either turn off 2FA completely, which is not advised or just for scheduled instructions, which is the preferred method.

To configure Tachyon not to require authentication for scheduled instructions open Tachyon Master database and locate a table called "GlobalSettings". In that table there will be a row with the name of "DefaultWorkflow". You have to examine the Value and change it.

By default, it looks like this:

DefaultWorkflow row value as configured by Tachyon installation

```
[
    {
        "ReferenceType": 0,
        "InstructionWorkflow": [
            {
                "InstructionType": 0,
                "Workflow": {
                    "StateMachine": "State"
                }
            },
            {
                "InstructionType": 1,
                "Workflow": {
                    "StateMachine": "State with Authentication and Approval"
                }
            }
        ]
    },
    {
        "ReferenceType": 2,
        "Workflow": {
            "StateMachine": "Scheduled Instruction State with Authentication and Approval",
            "Authenticating": "Scheduled Instruction Authentication",
            "Approval": "Scheduled Instruction Approval",
            "InProgress": "Scheduled Instruction InProgress",
            "Cancelling": "Scheduled Instruction Cancelling"
        }
    },
    {
        "ReferenceType": 3,
        "Workflow": {
            "StateMachine": "Agent Deployment State",
            "Processing": "Agent Deployment Processing",
            "Completing": "Agent Deployment Completing",
            "Cancelling": "Agent Deployment Cancelling"
        }
    },
    {
        "ReferenceType": 4,
        "Workflow": {
            "StateMachine": "Management Group Sync State",
            "Processing": "Management Group Sync Processing",
            "Completing": "Management Group Sync Completing"
        }
    },
    {
        "ReferenceType": 5,
        "Workflow": {
            "StateMachine": "Policy Deployment State",
            "Processing": "Policy Deployment Processing",
            "Completing": "Policy Deployment Completing"
        }
    }
]
```

Locate the JSON entry that reads:

`"StateMachine": "Scheduled Instruction State with Authentication and Approval"`

and change it to:

`"StateMachine": "Scheduled Instruction State with Approval".`

Save the modified value into the database.

Create a new schedule

In this section we will learn how to create a schedule.

When designig your schedule you have to take into account that a schedule can have two intervals. The main one, described by "ScheduleFreqType" and "ScheduleFreqInterval" can be supplemented by an "inner" one, each one occuring within a single "tick" of the main interval. This "inner" interval is described by "ScheduleFreqSubdayType" and ""ScheduleFreqSubdayInterval" properties.

This means that in some cases, there are multiple ways of creating effectively the same schedule. For example, if you wanted to create a schedule that executes daily at a given time, you could define a schedule, where the main schedule is set to daily and the inner one defiens a single execution at a given point in time, but you could also define a schedule where the main schedule is set to weekly and the inner one is set to executing once a day on every day of the week.

The "daily" version would look like this:

Schedule that executes an instruction once a day - "daily" version

```
{
    "Comments": null,
    "DefinitionName": "1E-Explorer-TachyonCore-AllInstalledSoftware",
    "InstructionTtlMinutes": 1440,
    "KeepRaw": 1,
    "Parameters": null,
    "ResponseTtlMinutes": 1440,
    "ScheduleActiveStartDate": "2020-2-15T00:00:00Z",
    "ScheduleActiveStartTime": 500,
    "ScheduleActiveEndDate": "2025-2-12T00:00:00Z",
    "ScheduleActiveEndTime": null,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 4,
    "ScheduleFreqInterval": 1,
    "ScheduleFreqSubdayType": 1,
    "ScheduleFreqSubdayInterval": null,
    "ScheduleFreqRelativeInterval": null,
    "ScheduleFreqRecurrenceFactor": null,
}
```

While the "weekly" version would look like this:

Schedule that executes an instruction once a day - "weekly" version

```
{
    "Comments": null,
    "DefinitionName": "1E-Explorer-TachyonCore-AllInstalledSoftware",
    "InstructionTtlMinutes": 1440,
    "KeepRaw": 1,
    "Parameters": null,
    "ResponseTtlMinutes": 1440,
    "ScheduleActiveStartDate": "2020-2-15T00:00:00Z",
    "ScheduleActiveStartTime": 500,
    "ScheduleActiveEndDate": "2025-2-12T00:00:00Z",
    "ScheduleActiveEndTime": 235900,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 8,
    "ScheduleFreqInterval": 127,
    "ScheduleFreqSubdayType": 1,
    "ScheduleFreqSubdayInterval": 0,
    "ScheduleFreqRelativeInterval": 0,
    "ScheduleFreqRecurrenceFactor": 1,
}
```

Finally, let's look at which API endpoint and SDK method to use when creating a schedule. I'll pick the "daily" one for the example.

Direct Consumer API call	C# code using Consumer SDK library
By making a POST request to https://my.tachyon.server/Consumer/ScheduledInstructions you can schedule an instruction using instruction definition's name:	Use ScheduledInstructions object inside the Tachyon connector instance.

Request payload

```
{
    "Comments": null,
    "DefinitionName": "1E-Explorer-TachyonCore-AllInstalledSoftware",
    "InstructionTtlMinutes": 1440,
    "KeepRaw": 1,
    "Parameters": null,
    "ResponseTtlMinutes": 1440,
    "ScheduleActiveStartDate": "2020-2-15T00:00:00Z",
    "ScheduleActiveStartTime": 500,
    "ScheduleActiveEndDate": "2025-2-12T00:00:00Z",
    "ScheduleActiveEndTime": null,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 4,
    "ScheduleFreqInterval": 1,
    "ScheduleFreqSubdayType": 1,
    "ScheduleFreqSubdayInterval": null,
    "ScheduleFreqRelativeInterval": null,
    "ScheduleFreqRecurrenceFactor": null,
}
```

or by using instruction definition's Id:

Request payload

```
{
    "Comments": null,
    "DefinitionId": 44,
    "InstructionTtlMinutes": 1440,
    "KeepRaw": 1,
    "Parameters": null,
    "ResponseTtlMinutes": 1440,
    "ScheduleActiveStartDate": "2020-2-15T00:00:00Z",
    "ScheduleActiveStartTime": 500,
    "ScheduleActiveEndDate": "2025-2-12T00:00:00Z",
    "ScheduleActiveEndTime": null,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 4,
    "ScheduleFreqInterval": 1,
    "ScheduleFreqSubdayType": 1,
    "ScheduleFreqSubdayInterval": null,
    "ScheduleFreqRelativeInterval": null,
    "ScheduleFreqRecurrenceFactor": null,
}
```

The return payload will contain information about the instruction we just scheduled:

Scheduling an instruction to run daily

```
var payload = new
ScheduledInstruction
{
    DefinitionName =
"1E-Explorer-
TachyonCore-
AllInstalledSoftware"
,

InstructionTtlMinutes
= 1440,
    KeepRaw = true,
    Parameters =
null,

ResponseTtlMinutes =
1440,

ScheduleActiveStartDa
te = new DateTime
(2020, 2, 15, 0, 0,
0),

ScheduleActiveStartTi
me = 500,

ScheduleActiveEndDate
= new DateTime(2025,
2, 12, 0, 0, 0),

ScheduleActiveEndTime
= null,
    ScheduleEnabled
= true,
    ScheduleFreqType
= 4,

ScheduleFreqInterval
= 1,

ScheduleFreqSubdayTyp
e = 1,

ScheduleFreqSubdayInt
erval = null,

ScheduleFreqRelativeI
nterval = null,

ScheduleFreqRecurrenc
eFactor = null,
};
var
scheduledInstruction
= connector.
ScheduledInstructions
.
SendScheduledInstruct
ion(payload);
```

"scheduledInstruction" object will contain the same data you can see in the JSON response on the left.

Return payload


```
{
    "Id": 9,
    "Name": "1E-Explorer-TachyonCore-AllInstalledSoftware",
    "Cmd": "SendAll",
    "KeepRaw": true,
    "Scope": null,
    "ReadableScope": null,
    "InstructionTtlMinutes": 1440,
    "ResponseTtlMinutes": 1440,
    "CreatedTimestampUtc": "2020-02-12T08:37:36.9Z",
    "Export": false,
    "ExportLocation": null,
    "ParentInstructionId": null,
    "InstructionDefinitionId": 36,
    "CreatedBy": "SomeDomain\\Marius.Gage",
    "ResultsFilter": null,
    "PreviousResultsFilter": null,
    "Devices": null,
    "ConsumerId": 1,
    "ConsumerName": "MyCustomConsumer",
    "ConsumerCustomData": null,
    "ParameterJson": null,
    "OffloadResponses": false,
    "RequestedFor": null,
    "Comments": null,
    "ActionedBy": null,
    "ActionReason": null,
    "Status": 0,
    "StatusTimestampUtc": null,
    "Workflow": "{\"StateMachine\":\"Scheduled Instruction State with
Authentication and Approval\",\"Authenticating\":\"Scheduled Instruction
Authentication\",\"Approval\":\"Scheduled Instruction Approval\",\"InProgress\":
\"Scheduled Instruction InProgress\",\"Cancelling\":\"Scheduled Instruction
Cancelling\"}",
    "WorkflowState": 0,
    "ApprovalOffloaded": false,
    "ReadablePayload": null,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 4,
    "ScheduleFreqInterval": 1,
    "ScheduleFreqSubdayType": 1,
    "ScheduleFreqSubdayInterval": 0,
    "ScheduleFreqRelativeInterval": 0,
    "ScheduleFreqRecurrenceFactor": 0,
    "ScheduleActiveStartDate": "2020-02-15T00:00:00Z",
    "ScheduleActiveEndDate": "2025-02-12T00:00:00Z",
    "ScheduleActiveStartTime": 500,
    "ScheduleActiveEndTime": null,
    "ScheduleLastExecuted": null,
    "ScheduleNextExecution": "2020-02-15T00:05:00Z",
    "ScheduleReadableFrequency": "Occurs every 1 day(s) at 00:05:00 starting on
15/02/2020 ending on 12/02/2025"
}
```


ScheduleActiveStartTime and ScheduleActiveEndTime

These two properties for the in-day boundaries of a schedule and their value will have different meaning depending on how the schedule is set up.

If the "main" schedule is defined as a daily schedule, then you can define the number of repeats each day. Those repeats will happen between **ScheduleActiveStartTime** and **ScheduleActiveEndTime** time.

If the "main" schedule is not a daily one, then **ScheduleActiveStartTime** will define the time of day on which the schedule should be activated.

Narrowing the set of devices that should respond to the scheduled instruction

Much like with regular instructions, scheduled ones can be scoped and targeted. The scope expression or targer list will be applied to each instruction issued on the schedule.

Issue an instruction, track its progress and retrieve responses describes how scoping and targeting works for instructions. To add scope you'd simply add **Scope** property to the schedule creating payload and you would build an expression defining a scope, exactly like you would for regular instruction.

For targeted schedule instructions, you would include the Devices array in the payload and instead of POSTing the payload to https://my.tachyon.server/Consumer/ScheduledInstructions you would POST to https://my.tachyon.server/Consumer/ScheduledInstructions/Targeted. If you're using the Consumer SDK, you would call connector.ScheduledInstructions.SendTargetedScheduledInstruction() instead of connector.ScheduledInstructions.SendScheduledInstruction().

Other instruction properties

As you have seen on Issue an instruction, track its progress and retrieve responses, instructions have other properties, like Parameters, ResultsFilters, Export options etc. These properties are the same on the scheduled instructions and what you set for the schedule will be copied into each instance of instruction issued by the schedule.

Modifying and cancelling a schedule

An existing schedule can be modified by using a PUT verb, instead or POST while calling either https://my.tachyon.server/Consumer/ScheduledInstructions or https://my.tachyon.server/Consumer/ScheduledInstructions/Targeted. You have to include the schedule Id (in the example above the schedule we created had an Id of 9) in the payload as well. If you're using the Consumer SDK, instead or using connector.ScheduledInstructions.SendTargetedScheduledInstruction() and connector.ScheduledInstructions.SendScheduledInstruction() you would use connector.ScheduledInstructions.UpdateScheduledInstruction() and connector.ScheduledInstructions.UpdateTargetedScheduledInstruction() respectively:

Direct Consumer API call	C# code using Consumer SDK library
PUT request to https://my.tachyon.server/Consumer/ScheduledInstructions: **Request payload** `{ "Id": 9, "Comments": null, "DefinitionName": "1E-Explorer-TachyonCore-AllInstalledSoftware", "InstructionTtlMinutes": 1440, "KeepRaw": 1, "Parameters": null, "ResponseTtlMinutes": 1440, "ScheduleActiveStartDate": "2020-2-15T00:00:00Z", "ScheduleActiveStartTime": 500, "ScheduleActiveEndDate": "2025-2-12T00:00:00Z", "ScheduleActiveEndTime": null, "ScheduleEnabled": true, "ScheduleFreqType": 4, "ScheduleFreqInterval": 1, "ScheduleFreqSubdayType": 1, "ScheduleFreqSubdayInterval": null, "ScheduleFreqRelativeInterval": null, "ScheduleFreqRecurrenceFactor": null, }` will yield result we've already seen when creating the schedule	Use ScheduledInstructions object inside the Tachyon connector instance. **Updating a schedule** `var payload = new ScheduledInstruction { DefinitionName = "1E-Explorer-TachyonCore-AllInstalledSoftware", InstructionTtlMinutes = 1440, KeepRaw = true, Parameters = null, ResponseTtlMinutes = 1440, ScheduleActiveStartDate = new DateTime(2020, 2, 15, 0, 0, 0), ScheduleActiveStartTime = 500, ScheduleActiveEndDate = new DateTime(2025, 2, 12, 0, 0, 0), ScheduleActiveEndTime = null, ScheduleEnabled = true, ScheduleFreqType = 4, ScheduleFreqInterval = 1, ScheduleFreqSubdayType = 1, ScheduleFreqSubdayInterval = null, ScheduleFreqRelativeInterval = null, ScheduleFreqRecurrenceFactor = null, }; var scheduledInstruction = connector.ScheduledInstructions.UpdateScheduledInstruction(9, payload);` "scheduledInstruction" object will contain the same data we've seen in the example where we created the schedule

To cancel a schedule all you need is its Id

Direct Consumer API call	C# code using Consumer SDK library

Making a DELETE request to https://my.tachyon.server/Consumer/ScheduledInstructions/9 will result in the schedule with the Id of 9 being deleted.	Use ScheduledInstructions object inside the Tachyon connector instance. **Cancelling a schedule** `connector.ScheduledInstructions.CancelSchedule(9);`

Example schedules

In this section we will look at several schedules. We will focus on the scheduling properties of the payload for brevity and assume each schedule issues the same instruction. Please remember that all times are in UTC.

First let's look at a schedule that executes just once, at 19:30 on 18th of February 2020.

Execute once schedule

```
{
   "ScheduleActiveStartDate": "2020-2-18T00:00:00Z",
   "ScheduleActiveStartTime": 193000,
   "ScheduleActiveEndDate": null,
   "ScheduleActiveEndTime": null,
   "ScheduleEnabled": true,
   "ScheduleFreqType": 1,
   "ScheduleFreqInterval": 0,
   "ScheduleFreqSubdayType": 0,
   "ScheduleFreqSubdayInterval": 0,
   "ScheduleFreqRelativeInterval": 0,
   "ScheduleFreqRecurrenceFactor": 0
}
```

Now let's look at a more complicated use case. Imagine if you will that you have a task that requires user interaction and should be completed within a week. Although it requires user interaction, you can still check if it has been done via an instruction, and you want to get the notification that a task has been done within the hour if possible. You decide to to create a schedule where the check instruction is executed every day, repeated every hour within business hours (9 to 17), as there is no point in checking when there's no one in the office.

Execute once a day for a week, 8 times a day

```
{
    "ScheduleActiveStartDate": "2020-2-22T00:00:00Z",
    "ScheduleActiveStartTime": 90000,
    "ScheduleActiveEndDate": "2020-2-29T00:00:00Z",
    "ScheduleActiveEndTime": 170000,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 4,
    "ScheduleFreqInterval": 1,
    "ScheduleFreqSubdayType": 8,
    "ScheduleFreqSubdayInterval": 1,
    "ScheduleFreqRelativeInterval": 0,
    "ScheduleFreqRecurrenceFactor": 0
}
```

This time let's say you'd like to, on a Friday evening (18:00) run an instruction that, for instance, checks what new software was installed in the last 7 days. To do that you could create a weekly schedule that executes only on Friday at 18:00:

Execute once a week on Friday at 18:00

```
{
    "ScheduleActiveStartDate": "2020-2-22T00:00:00Z",
    "ScheduleActiveStartTime": 180000,
    "ScheduleActiveEndDate": "2025-2-15T00:00:00Z",
    "ScheduleActiveEndTime": 235900,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 8,
    "ScheduleFreqInterval": 32,
    "ScheduleFreqSubdayType": 1,
    "ScheduleFreqSubdayInterval": 0,
    "ScheduleFreqRelativeInterval": 0,
    "ScheduleFreqRecurrenceFactor": 1
}
```

Lastly, let's look at a schedule that runs an instruction once a month, on the 7th day of the month at 9:00.

Execute once a month on the 7th day of the month at 9:00

```
{
    "ScheduleActiveStartDate": "2020-2-24T00:00:00Z",
    "ScheduleActiveStartTime": 90000,
    "ScheduleActiveEndDate": "2025-2-24T00:00:00Z",
    "ScheduleActiveEndTime": null,
    "ScheduleEnabled": true,
    "ScheduleFreqType": 16,
    "ScheduleFreqInterval": 7,
    "ScheduleFreqSubdayType": 1,
    "ScheduleFreqSubdayInterval": 0,
    "ScheduleFreqRelativeInterval": 0,
    "ScheduleFreqRecurrenceFactor": 1
}
```