Add and remove Instruction Definitions and organize them into Instruction Sets

Introduction

This section covers uploading instruction definitions into Tachyon and organizing them into sets.

Instruction Definitions

Instruction definitions are templates, used by Tachyon to issue Instructions. Before using an instruction definition it has to be uploaded and assigned to an instruction set.

On this page:

Uploading Instruction Definitions

Instruction definitions can be uploaded individually, one XML at a time, or in bulk as a zip file containing multiple instruction definitions:

When using this method to POST an instruction, the format of the POST payload is a MIME multipart document. The C# SDK contains helper functions to handle this automatically. If you are working in another environment, such as PowerShell, then the payload format looks like this for a raw, uncompressed XML file

1. You must add a header to the request indicating the MIME type and boundary to the POST request.

```
Content-Type: multipart/form-data; boundary=xyzzy
```

(where xyzzy is an arbitrary delimiter string you do not expect to find in your payload content). A GUID is a good choice

2. The payload is then constructed by concatenating the MIME multipart header as shown below. In the sample, the uploaded filename is x.xml

```
--xyzzy

Content-Disposition: form-data; name="x.xml"; filename="x.xml"
Content-Type: text/xml

<the XML that corresponds to the signed instruction i.e contents of the XML file>
--xyzzy--
```

Note that the MIME delimiter begins with two hyphens at the start and has two hypens at the front AND end as the closing delimiter

For a zipped file the content will be set to application/zip

Direct Consumer API call	C# code using Consumer SDK library

Make a POST request to https://my.tachyon.server/Consumer/ ProductPacks attach the file (either XML or zip) to the request.

The return payload contains information about instruction definitions found in the file and at the status of the upload.

Return payload

```
[{
    "InstructionDefinitionName": "1E-Explorer-
Examples-CountEventLogErrorsInLastX",
    "InstructionSetName": "Unassigned",
    "Status": "New"
},
{
    "InstructionDefinitionName": "1E-Explorer-
Examples-DeleteCertificates",
    "InstructionSetName": "Unassigned",
    "Status": "New"
}]
```

Use ProductPacks object inside the Tachyon connector instance.

Uploading instruction definitions

```
response = connector.ProductPacks.Upload("c:
\\product packs\\1E-ConfigMgrConsoleExtensions.zip");
```

response.ReceivedObject object contains the same data you can see in the JSON response on the left.

Upload instruction definitions into an existing instruction set:

Direct Consumer API call	C# code using Consumer SDK library
Make a POST request to https://my.tachyon.server/Consumer/ProductPacks /InstructionSet/Id/1 and attach the file (either XML or zip) to the request. The return payload will (as in the above example) contain information about instruction definitions found in the file and at the status of the upload. **Return payload** `[{ "InstructionDefinitionName": "1E-Explorer-Examples-CountEventLogErrorsInLastX", "InstructionSetName": "Hardware questions"", "Status": "New" }, { "InstructionDefinitionName": "1E-Explorer-Examples-DeleteCertificates", "InstructionSetName": "Hardware questions"", "Status": "New" }]`	Use ProductPacks object inside the Tachyon connector instance. **Uploading instruction definitions** `response = connector.ProductPacks. UploadIntoInstructionSet("c:\\product packs\\1E-ConfigMgrConsoleExtensions.zip", 1);` response.ReceivedObject object contains the same data you can see in the JSON response on the left.

Deleting Instruction Definitions

An instruction definition can be deleted if there are no pending, in progress or scheduled instructions based on it. If there are, either stop those instructions, let them run their course or delete the schedule.

Delete a single instruction definition, using either its Id or name:

Direct Consumer API call	C# code using Consumer SDK library

To delete an instruction definition using its Id make a DELETE request to https://my.tachyon.server/Consumer /InstructionDefinitions/Id/1
 	
To delete an instruction definition using its name make a DELETE request to https://my.tachyon.server/Consumer /InstructionDefinitions/ name/1E-Explorer-Examples-CountEventLogErrorsInLastX
	Use InstructionDefinitions object inside the Tachyon connector instance.
	
	Delete an Instruction Definition by its Id
	
	`connector.InstructionDefinitions.DeleteInstructionDefinition(1);`
	
	Delete an Instruction Definition by its Name
	
	`connector.InstructionDefinitions.DeleteInstructionDefinition("1E-Explorer-Examples-CountEventLogErrorsInLastX");`

Delete multiple instruction definitions in a single call using a list of either Ids or names:

Direct Consumer API call	C# code using Consumer SDK library

To delete multiple instruction definitions using their Ids make a DELETE request with following payload:
 	
Payload sent to https://my.tachyon.server/Consumer /InstructionDefinitions
 	
`{ "Ids": [ 1,2 ]}`
 	
To delete multiple instruction definitions using their names make a DELETE request with following payload:
 	
Payload sent to https://my.tachyon.server/Consumer /InstructionDefinitions
 	
`{ "Names": [ "1E-Explorer-Examples-CountEventLogErrorsInLastX", "1E-Explorer-Examples-DeleteCertificates" ]}`
	Use InstructionDefinitions object inside the Tachyon connector instance. To delete instruction definition using their Ids:
	
	Deleting multiple instruction definitions by their Ids
	
	`var payload = new List<int> { 1, 2 }; var response = connector.InstructionDefinitions.DeleteInstructionDefinitions(payload);`
	
	To delete instruction definitions using their names:
	
	Deleting multiple instruction definitions by their name
	
	`var payload = new List<string> { "1E-Explorer-Examples-CountEventLogErrorsInLastX", "1E-Explorer-Examples-DeleteCertificates" }; var response = connector.InstructionDefinitions.DeleteInstructionDefinitions(payload);`

Instruction Sets

Instruction sets are containers holding instruction definitions:

- With permissions assigned to instruction sets, not to individual instructions
- Containing any number of instruction definitions and cannot be nested
- Instruction definitions can only be assigned to one instruction set, or none at all.

An instruction definition cannot belong to multiple instruction sets at once.

It also does not have to belong to a set. If an instruction definition does not belong to an instruction set, you cannot issue instructions based on that definition .

Add an Instruction Set

To add an instruction set all you need is a name. This name needs to be unique and no two instruction sets can have the same name.

Direct Consumer API call	C# code using Consumer SDK library

Making a POST request with following payload:
 	
Payload sent to https://my.tachyon.server/Consumer/InstructionSets
 	
`{`
`  "Name": "Hardware questions"`
`}`
 	
Returns the following response:
 	
Return payload
 	
`{`
`  "Id": 1,`
`  "Name": "Hardware questions",`
`  "Description": null,`
`  "Icon": null`
`}`
	Use InstructionSets object inside the Tachyon connector instance.
	
	Creating an Instruction Set
	
	`var payload = new Tachyon.SDK.Consumer.Models.Send.InstructionSet`
	`{`
	`  Name = "Hardware questions"`
	`};`
	
	`var response = connector.InstructionSets.Add(payload);`
	
	response.ReceivedObject object will contain the same data you can see in the JSON response on the left.

An instruction set can also have a description and an icon.

The icon can be supplied either directly as binary data inside the Icon parameter or as the same binary data encoded in a base64 string supplied in IconInBase64 parameter.

UseIcon parameter is a flag indicating if an icon should be used.

If this flag is set to true, icon will be used in following way, if:

- Icon field (byte array) is supplied, it will be used as the icon
- Icon isn't supplied and IconInBase64 is supplied, it will be used as the icon
- both Icon and IconInBase64 are supplied, Icon will be used and IconInBase64 will be ignored
- Neither Icon or IconInBase64 were supplied, icon will not be used and if there already was an icon assigned to the instruction set, it will be removed.

If this flag is set to false, no changes will be made to the icon. Both Icon and IconInBase64 fields are ignored.

To create an Instruction Set with a description and an icon:

Direct Consumer API call	C# code using Consumer SDK library

Making a POST request with following payload:

Payload sent to https://my.tachyon.server/Consumer/InstructionSets

```
{
  "Name": "Hardware questions",
  "Description": "This is a description",
  "UseIcon": true,
  "Icon": [<binary data here>]
}
```

will create an instruction set that uses binary data inside Icon. In order to use binary data encoded as base64 string make POST request with following payload:

Payload sent to https://my.tachyon.server/Consumer/InstructionSets

```
{
  "Name": "Hardware questions",
  "Description": "This is a description",
  "UseIcon": true,
  "IconInBase64": "<binary data encoded as
base64 here>"
}
```

Both POST calls will yield following response because the icon is always returned in its binary form:

Return payload

```
{
    "Id": 1,
    "Name": "Hardware questions",
    "Description": "This is a description",
    "Icon": [<binary data here>]
}
```

Use InstructionSets object inside the Tachyon connector instance.

To create an instruction set that uses an icon supplied as a binary array you can use code below and supply the icon inside binaryData variable.

Creating an Instruction Set

```
var payload = new Tachyon.SDK.Consumer.Models.Send.
InstructionSet
{
    Name = "Hardware questions",
    Description = "This is a description",
    UseIcon = true,
    Icon = binaryData
};

var response = connector.InstructionSets.Add
(payload);
```

To create an instruction set that uses an icon supplied as binary data encoded in a base64 string, you can use code below and supply the icon inside encodedData variable:

Creating an Instruction Set

```
var payload = new Tachyon.SDK.Consumer.Models.Send.
InstructionSet
{
    Name = "Hardware questions",
    Description = "This is a description",
    UseIcon = true,
    IconInBase64 = encodedData
};

var response = connector.InstructionSets.Add
(payload);
```

In either case, response.ReceivedObject object will contain the same data you can see in the JSON response on the left.

Edit an Instruction Set

To edit an instruction set, you'll need its Id.

You can modify all properties of an instruction set, apart from the Id. Icon follows the same rules as outlined in the previous paragraph. This means the payload looks similar to that used to create an instruction set, with the addition of Id, and making PUT requests instead of POST.

Direct Consumer API call	C# code using Consumer SDK library

Making a PUT request with following payload:

Payload sent to https://my.tachyon.server/Consumer/InstructionSets

```
{
  "Id": 1,
  "Name": "Hardware instructions"
}
```

Returns this response providing the instruction set with the Id of 1 didn't have an icon:

Return payload

```
{
    "Id": 1,
    "Name": "Hardware instructions",
    "Description": null,
    "Icon": null
}
```

Use InstructionSets object inside the Tachyon connector instance.

Updating an Instruction Set

```
var payload = new Tachyon.SDK.Consumer.Models.Send.
InstructionSet
{
   Id = 1,
   Name = "Hardware instructions"
};

var response = connector.InstructionSets.Update(payload);
```

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

Delete an Instruction Set

An instruction set can be deleted, causing all instruction definitions belonging to it to become unassigned.

Direct Consumer API call	C# code using Consumer SDK library
Make a DELETE request to https://my.tachyon.server/Consumer/ InstructionSets /Id/ 1	Use InstructionSets object inside the Tachyon connector instance. **Deleting an Instruction Set** `connector.InstructionSets.Delete(1);`

You can also delete an instruction set along with all instruction definitions inside that set. Use this option with care as it permanently removes the instruction definitions.

Direct Consumer API call	C# code using Consumer SDK library
Make a DELETE request to https://my.tachyon.server/Consumer/ InstructionSets/Id/ 1?deleteContent=true	Use InstructionSets object inside the Tachyon connector instance. **Deleting an Instruction Set with all Instruction Definitions in that set** `connector.InstructionSets.DeleteWithContents(1, true);`

You cannot delete Instruction Definitions with pending, in progress Instructions or scheduled Instructions, so trying to delete an Instruction Set with contents with instruction definitions like these will fail.

Get Instruction Sets

The first example shows how to retrieve all available instruction sets:

Direct Consumer API call	C# code using Consumer SDK library

Making a GET request to https://my.tachyon.server/Consumer/ InstructionSets returns this response:

Return payload

```
[{
    "Id": 1,
    "Name": "Hardware instructions",
    "Description": "This pack contains questions and
actions dealing with hardware",
    "Icon": null
},{
    "Id": 2,
    "Name": "Software instructions",
    "Description": "This pack contains questions and
actions dealing with software",
    "Icon": null
},{
    "Id": 3,
    "Name": "Patch success instructions",
    "Description": null,
    "Icon": null
}]
```

Use InstructionSets object inside the Tachyon connector instance:

Retrieving all Instruction Sets

```
response = connector.InstructionSets.GetAll();
```

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

You can also retrieve all instruction sets along with the count of instruction definitions that belong to each set:

Direct Consumer API call	C# code using Consumer SDK library

Making a GET request to https://my.tachyon.server/Consumer/ InstructionSets returns this response:

Return payload

```
[{
    "Id": 1,
    "Name": "Hardware instructions",
    "Description": "This pack contains questions and
actions dealing with hardware",
    "Icon": null,
    "CountOfDefinitions": 22
},{
    "Id": 2,
    "Name": "Software instructions",
    "Description": "This pack contains questions and
actions dealing with software",
    "Icon": null,
    "CountOfDefinitions": 37
},{
    "Id": 3,
    "Name": "Patch success instructions",
    "Description": null,
    "Icon": null,
    "CountOfDefinitions": 8
}]
```

Use InstructionSets object inside the Tachyon connector instance:

Retrieving all Instruction Sets

```
response = connector.InstructionSets.
GetAllWithCounts();
```

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

You can also retrieve a specific instruction set by its Id or Name:

Direct Consumer API call	C# code using Consumer SDK library

Making a GET request to https://my.tachyon.server/Consumer/ InstructionSets/Id/1 returns this response:

Return payload

```
{
    "Id": 1,
    "Name": "Hardware questions",
    "Description": null,
    "Icon": null
}
```

Use InstructionSets object inside the Tachyon connector instance:

Retrieving specific Instruction Set

```
response = connector.InstructionSets.Get(id);
```

response.ReceivedObject object will contains the same data as the JSON response on the left.

Add Instruction Definitions to an Instruction Set

Instruction definitions can be added to an Instruction set either using the Instruction Set API or Instruction Definitions API.

Using the Instruction Set API:

Direct Consumer API call	C# code using Consumer SDK library

Making a POST request with following payload:
 	
Payload sent to https://my.tachyon.server/Consumer /InstructionSets/Contents
 	
`{ "SetId": 1, "InstructionDefinitionIds": [ 1,2,3 ] }`
 	
returns this response:
 	
Return payload
 	
`{ "Id": 1, "Name": "Hardware questions", "Description": null, "Icon": null }`
	Use InstructionSets object inside the Tachyon connector instance.
	
	Adding Instruction Definitions to specific Instruction Set
	
	`response = connector.InstructionSets. AddInstructionDefinitions(1, new List<int> {1,2,3});`
	
	response.ReceivedObject object contains the same data you as the JSON response on the left.

If you do not provide SetId in your payload, the call will succeed but the instructions will become unassigned.

Using Instruction Definition API:

Direct Consumer API call	C# code using Consumer SDK library
Making a POST request to https://my.tachyon.server/Consumer /InstructionDefinitions/Id/1 /InstructionSet/1 without any payload returns this response: **Return payload** `{ "Id": 1, "Name": "1E-Explorer-Examples-CountEventLogErrorsInLastX", "Description": "Returns event log entries aggregated on event ID (EventCode) of a specified type in the a specified Windows event log in the last few days.`	

```
Specifying a long period may result in a large amount of data from
each
agent.",
    "InstructionSetId": 2,
    "InstructionSetName": "Wszystko",
    "InstructionType": 0,
    "ReadablePayload": "Which event IDs of type %type% have been
logged in the Windows %logname% event log in the last %numDays%
days?",
    "Parameters": [{
        "Name": "type",
        "Pattern": "%type%",
        "DataType": "string",
        "ControlType": "valuePicker",
        "ControlMetadata": null,
        "Placeholder": "event type",
        "DefaultValue": null,
        "Validation": {
            "Regex": null,
            "MaxLength": null,
            "AllowedValues": ["information",
            "warning",
            "error"],
            "NumValueRestrictions": null
        },
        "Value": null,
        "HintText": null,
        "Source": null
    },
    {
        "Name": "logname",
        "Pattern": "%logname%",
        "DataType": "string",
        "ControlType": "valuePicker",
        "ControlMetadata": null,
        "Placeholder": "log name'",
        "DefaultValue": null,
        "Validation": {
            "Regex": null,
            "MaxLength": null,
            "AllowedValues": ["application",
            "security",
            "setup",
            "system"],
            "NumValueRestrictions": null
        },
        "Value": null,
        "HintText": null,
        "Source": null
    },
    {
        "Name": "numDays",
        "Pattern": "%numDays%",
        "DataType": "int",
        "ControlType": "freeText",
        "ControlMetadata": null,
        "Placeholder": "number of days",
        "DefaultValue": null,
        "Validation": {
            "Regex": null,
            "MaxLength": null,
            "AllowedValues": null,
            "NumValueRestrictions": null
        },
        "Value": null,
        "HintText": null,
        "Source": null
    }],
    "Schema": [{
        "Name": "Category",
        "Type": "int64",
```


Use InstructionDefinitions object inside the Tachyon connector instance.

Addign specific Instruction Definition to an Instruction Set

```
response = connector.
InstructionDefinitions.
AddDefinitionToSet(1, 1);
```

response.ReceivedObject object contains the same data in the JSON response on the left.

```
        "Length": 0,
        "RenderAs": null
    },
    {
        "Name": "CategoryString",
        "Type": "String",
        "Length": 256,
        "RenderAs": null
    },
    {
        "Name": "EventCode",
        "Type": "int64",
        "Length": 0,
        "RenderAs": null
    },
    {
        "Name": "EventIdentifier",
        "Type": "int64",
        "Length": 0,
        "RenderAs": null
    },
    {
        "Name": "Message",
        "Type": "String",
        "Length": 4096,
        "RenderAs": null
    },
    {
        "Name": "RecordNumber",
        "Type": "int64",
        "Length": 0,
        "RenderAs": null
    },
    {
        "Name": "SourceName",
        "Type": "String",
        "Length": 256,
        "RenderAs": null
    },
    {
        "Name": "GeneratedTimeStamp",
        "Type": "datetime",
        "Length": 0,
        "RenderAs": null
    },
    {
        "Name": "User",
        "Type": "String",
        "Length": 256,
        "RenderAs": null
    }],
    "Aggregation": {
        "Schema": [{
            "Name": "EventCode",
            "Type": "int64",
            "Length": 0,
            "RenderAs": null
        },
        {
            "Name": "Count",
            "Type": "int32",
            "Length": 0,
            "RenderAs": null
        }],
        "GroupBy": "EventCode",
        "Operations": [{
            "Name": "Count",
            "Type": "count"
        }]
    },
    "InstructionTtlMinutes": 30,
```

```
    "ResponseTtlMinutes": 30,
    "MinimumInstructionTtlMinutes": 10,
    "MaximumInstructionTtlMinutes": 10080,
    "MinimumResponseTtlMinutes": 10,
    "MaximumResponseTtlMinutes": 10080,
    "Workflow": null,
    "ResponseTemplateId": 1,
    "OriginalFileName": null,
    "ResponseTemplateConfiguration": {
        "Name": "default",
        "TemplateConfigurations": [{
            "Id": "leftpie",
            "Title": "Number of EventIds across devices",
            "Type": "Pie",
            "X": "EventCode",
            "Y": "Count",
            "PostProcessor": "processingFunction",
            "Size": 1,
            "Row": 1
        }],
        "PostProcessors": [{
            "Name": "processingFunction",
            "Function": "ProcessSingleSeries('EventCode', 'Count',
'9', 'Number of events', 'true')"
        }]
    },
    "Version": "3",
    "Author": "1E",
    "IsLicensed": true,
    "UploadedTimestampUtc": "2019-03-19T10:14:32.227Z",
    "NumberOfTimesExecuted": 0,
    "Executable": true
}
```

The InstructionSet API allows you to bulk assign Definitions, while InstructionDefinitions API allows you to only assign one definition to a set at a time.

Remove Instruction Definitions from an Instruction Set

Instruction definitions can be removed from an Instruction set either using the Instruction Set API or Instruction Definitions API.

Using Instruction Set API:

Direct Consumer API call	C# code using Consumer SDK library
Make a DELETE request with following payload: **Payload sent to https://my.tachyon.server/Consumer/InstructionSets/Contents** `{ "SetId": 1, "InstructionDefinitionIds": [ 1,2,3 ] }`	Use InstructionSets object inside the Tachyon connector instance: **Removing Instruction Definitions from specific Instruction Set** `connector.InstructionSets.RemoveInstructionDefinitions(1, new List<int> { 1, 2, 3 });`

Using Instruction Definition API.

Direct Consumer API call	C# code using Consumer SDK library

Make a DELETE request to https://my.tachyon.server/Consumer /InstructionDefinitions/Id/1/InstructionSet/1	Use InstructionDefinitions object inside the Tachyon connector instance: **Removing a specific Instruction Definition from specific Instruction Set** `connector.InstructionDefinitions.RemoveDefinitionFromSet(1, 1);`

Export Instruction Definitions from a set

The following example exports all instruction definitions that belong to a specific set, allowing you to download an entire Instruction Set without knowing which instruction definitions belong to that set.

Direct Consumer API call	C# code using Consumer SDK library
Making a GET request to https://my.tachyon.server/Consumer /InstructionSets/Id/{id}/Export will give you back a binary stream with a zip archive that contains all Instruction Definitions from the requested set.	Use InstructionSets object inside the Tachyon connector instance: **Exporting all Instruction Definitions that belong to a specific Instruction Set** `response = connector.InstructionSets.Export(1);` response.ReceivedObject will be a MemoryStream holding a zip file containing Instruction Definitions.