Find the right instruction definition

Introduction

This section covers how to browse available instruction definitions in your Tachyon installation.

In the following examples, the instruction definitions have already been uploaded into Tachyon, organized into Instruction Sets and have assigned permissions . For more information about this read Add and remove Instruction Definitions and organize them into Instruction Sets and Set up Principals, Roles and Permissions .

The C# examples assume you're using Tachyon Consumer SDK and you have an 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 is always true .

When this section mentions permissions on instructions, it means i nstruction set permissions not Instructi on set management permissions . Instruction set management permissions are a separate category of permissions. They gain Principals access to all instructions within a specific set or all sets.

It's possible to have a permission on specific, or every instruction set and be able to issue those instructions without being able to manage their instruction set (that is, without having associated instruction set management permissions) .

The reverse is also possible and the Principal that can assign instruction definitions to an instruction set may not have the necessary permissions to issue any of them .

On this page:

What is an instruction definition?

An instruction definition is a template used to create instructions which are sent to computers running Tachyon Agent for processing.

The instruction definition defines what the agent does when executing the instruction, for example what data is returned, if it is aggregated, what the parameters for the instruction are, or how to present the results to the user.

When an instruction is issued, some information is copied from the definition the instruction is based on (like instruction name and type) while others come from the information provided with the request to issue the instruction. For more information read issue an instruction, track its progress and retrieve responses.

What properties can be used to uniquely identify an instruction definition?

An instruction definition has two properties to identify it, Id and name:

Id - is a number assigned to the instruction definition when it's imported into Tachyon. This means you cannot predict the Id before the instruction is imported and it will be different between Tachyon installations. If you remove the instruction definition and import it again it will be given a different Id.

Name - must be unique in each Tachyon installation. Importing a new instruction definition with the same name as one already in Tachyon is treated as an upgrade of the existing instruction definition.

The API allows you to use both Id and Name when locating specific instruction definitions or issuing an instruction.

How to uniquely identify an instruction?

Instructions can only be identified by their Id, which is assigned when an instruction is created (issued). They cannot be located by Name, because it's not unique. Every instruction based on the same instruction definition will have the same name.

How is an instruction created from an instruction definition

Full details of this process are outside the scope of this section.

When an instruction is created (sometimes called issuing) the:

- Instruction definition name gets copied into the newly created instruction

- The payload and readable payload get copied into the newly created instruction - any parameters (provided by the issuer of the instruction), environment variables and global settings are completed
- Instruction TTL and Response TTL are filled with values provided by the instruction issuer
- Schema and Aggregation schema get copied from the instruction definition
- Scope provided by the instruction issuer is combined with any compulsory scope that results from issuing Principal permissions.

This is not a complete list.

Searching for instruction definitions

Instruction definitions are retrieved for two reasons:

Issuing instructions - instruction set related permissions of the Principal have to be taken into account. Those permissions limit which instructions the caller is allowed to issue. This limitation is enforced by the API endpoint.

Managing instruction sets - the Principal retrieving them requires permission to manage instruction sets and does not need to have any permissions to the instructions themselves.

This is important because if you retrieve all instruction definitions using the endpoint designed for instruction set management, you might not be able to issue all of those instructions. This is because the list you receive will not be filtered down using your permissions.

Retrieving all instruction definitions

This example retrieves all instruction definitions that can be issued:

Direct Consumer API call	C# code using Consumer SDK library

Making a GET request to https://my.tachyon.server/Consumer/Instruction Definitions

Response received from https://my.tachyon.server/Consumer/InstructionDefinitions


```
[{
    "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",
        "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
```

Use InstructionDefinitions object inside the Tachyon connector instance.

Retrieving all instruction definitions the caller has permissions to

```
var definitions = connector.InstructionDefinitions.Get(null);
```

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

```
    },
    {
        "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
```

```
},
{
    "Id": 2,
    "Name": "1E-Explorer-Examples-DeleteCertificates",
    "Description": "Deletes all certificates with a given SHA1 thumbprint in the 'local
machine' certificate store for these stores: Personal (My), Intermediate Certification
Authorities (CA), Trusted Root Certification Authorities (Root), Third-Party Certification
Authorities (AuthRoot), Enterprise Trust (Trust), Other People (AddressBook), Trusted People
(TrustedPeople), Trusted Publishers (TrustedPublisher) and Untrusted Certificates
(Disallowed). Certificates are not retrievable once marked for deletion. Please use with
caution - deleting a certificate may case a number of things to stop working.",
    "InstructionSetId": 2,
    "InstructionSetName": "Wszystko",
    "InstructionType": 1,
    "ReadablePayload": "Delete all certificates with a SHA1 thumbprint of %thumbprint% from
the 'local machine' certificate store.",
    "Parameters": [{
        "Name": "thumbprint",
        "Pattern": "%thumbprint%",
        "DataType": "string",
        "ControlType": "freeText",
        "ControlMetadata": null,
        "Placeholder": "thumbprint",
        "DefaultValue": null,
        "Validation": {
            "Regex": null,
            "MaxLength": "256",
            "AllowedValues": null,
            "NumValueRestrictions": null
        },
        "Value": null,
        "HintText": null,
        "Source": null
    }],
    "Schema": [{
        "Name": "Subject",
        "Type": "String",
        "Length": 256,
        "RenderAs": null
    },
    {
        "Name": "Thumbprint",
        "Type": "String",
        "Length": 128,
        "RenderAs": null
    },
    {
        "Name": "StoreName",
        "Type": "String",
        "Length": 256,
        "RenderAs": null
    },
    {
        "Name": "DeletionTime",
        "Type": "String",
        "Length": 256,
        "RenderAs": null
    }],
    "Aggregation": null,
    "InstructionTtlMinutes": 60,
    "ResponseTtlMinutes": 120,
    "MinimumInstructionTtlMinutes": 10,
    "MaximumInstructionTtlMinutes": 10080,
    "MinimumResponseTtlMinutes": 10,
    "MaximumResponseTtlMinutes": 10080,
    "Workflow": null,
    "ResponseTemplateId": null,
    "OriginalFileName": null,
    "ResponseTemplateConfiguration": null,
    "Version": "3",
    "Author": "1E",
```

```
    "IsLicensed": true,
    "UploadedTimestampUtc": "2019-03-19T10:14:32.227Z",
    "NumberOfTimesExecuted": 0,
    "Executable": true
}]
```

The payload has been trimmed to just two entries to show what you can expect. In a production installation of Tachyon there will be many more instruction definitions, perhaps hundreds.

Retrieving only instruction definitions of a specific type

In some cases, you may want to retrieve all instruction definitions of a given type that you can issue. This is used, for example, by Tachyon Explorer on the Follow-up question and Follow-up action tabs on the instruction details page. You can do this using the optional query string parameter on the same call as the one made in the last example.

The next example shows where we only get the Questions we're allowed to issue.

Direct Consumer API call	C# code using Consumer SDK library
Make a GET request to https://my.tachyon.server/Consumer/Instruction Definitions?instructionType=question The return payload will have the same structure as the payload seen when we retrieved all the instruction definitions in the previous example.	Use InstructionDefinitions object inside the Tachyon connector instance. **Retrieving all instruction definitions of a specific type that the caller is allowed to issue** `var definitions = connector.InstructionDefinitions.Get(new List<InstructionType>{InstructionType.Question});` Definitions object contains information similar to the retrieving all instruction definitions example.

Retrieving all instruction definitions regardless of caller permissions

Retrieving all definitions should only be used when managing instruction sets.

Direct Consumer API call	C# code using Consumer SDK library
Making a GET request to https://my.tachyon.server/Consumer/ InstructionDefinitions/All The return payload has same structure as the payload seen when retrieving all the instruction definitions in the retrieving all instruction definitions examples.	Use InstructionDefinitions object inside the Tachyon connector instance. **Retrieving all instruction definitions in the system, regardless of caller's permissions** `var definitions = connector.InstructionDefinitions.GetAllInstructionDefinitions();` Definitions object contains information similar to retrieving all instruction definitions examples.

Please note that this endpoint requires Instruction set management related permissions to work.

Using search-as-you-type feature

This is the feature seen on the Tachyon Explorer home page as "I want to know" in form of a search box where Tachyon returns the instruction definitions you have permissions to use based on your search criteria.

The match is achieved by searching for the words (defined as a string of characters separated by a space) in an instruction definition's readable payload (which is what Explorer displays after you've picked an instruction definition) and description (which Explorer displays in the tooltip available after an instruction definition is selected).

This example looks for the word bios and retrieves all instruction definitions containing that word either in the description or readable payload.

Direct Consumer API call	C# code using Consumer SDK library

Making a GET request to https://my.tachyon.server/Consumer /InstructionDefinitions/Search/Ymlvcw== returns results similar to:

Response received from https://my.tachyon.server/Consumer /InstructionDefinitions/Search


```
{
    "Result": 0,
    "Error": null,
    "Instructions": [
        {
            "Id": 66,
            "Name": "1E-Explorer-TachyonCore-
BiosDetails",
            "ReadableName": "What BIOS firmware is
installed?",
            "Description": "Returns details of BIOS
firmware.",
            "Category": "SomeCategory",
            "WordMatchCount": 1,
            "InstructionType": 0,
            "InstructionTtlMinutes": 60,
            "ResponseTtlMinutes": 60,
            "ParameterHints": null,
            "Aggregatable": true,
            "IsLicensed": true,
            "NumberOfTimesExecuted": 0,
            "Executable": null
        }
    ]
}
```


Use InstructionDefinitions object inside the Tachyon connector instance.

Retrieving instruction definitions matching a search term

```
var definitions = connector.
InstructionDefinitions.FindInstructions
("bios");
```

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

Entries for Instruction definitions returned by this endpoint contain only part of the data for performance reasons. This is intended as a search-as-you-type feature so the data must be returned quickly.

Searching for instruction definitions that match specific criteria

When managing instruction sets you can also use an endpoint allowing instruction definitions to be filtered, sorted and paginated. This endpoint, like the one allowing you to retrieve all instructions regardless of calling Principal's permissions, requires permissions related to instruction set management to work.

This example looks for all instruction definitions containing the word bios in their name. Although it looks the same as the search-as-you-type scenario, it looks exclusively on the Name, and not on the readable payload or description because our search parameters only specify the name.

Direct Consumer API call	C# code using Consumer SDK library
Making a POST request to https://my.tachyon.server/Consumer /InstructionDefinitions/Search with following payload:	

Payload sent to https://my.tachyon.server/Consumer/InstructionDefinitions /Search

```
{
  "Start": 1,
  "PageSize": 200,
  "Filter": {
    "Attribute": "Name",
    "Operator": "LIKE",
    "Value": "%bios%"
  }
}
```

Returns a similar response to the one below:

Return payload


```
{
    "TotalCount": 1,
    "Items": [
        {
            "Id": 66,
            "Name": "1E-Explorer-TachyonCore-BiosDetails",
            "Description": "Returns details of BIOS
firmware.",
            "InstructionSetId": 2,
            "InstructionSetName": "Wszystko",
            "InstructionType": 0,
            "ReadablePayload": "What BIOS firmware is
installed?",
            "Parameters": null,
            "Schema": [
                {
                    "Name": "BIOSVersion",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                },
                {
                    "Name": "Caption",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                },
                {
                    "Name": "Manufacturer",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                },
                {
                    "Name": "PrimaryBIOS",
                    "Type": "bool",
                    "Length": 0,
                    "RenderAs": null
                },
                {
                    "Name": "ReleaseDate",
                    "Type": "datetime",
                    "Length": 0,
                    "RenderAs": null
                },
                {
                    "Name": "SerialNumber",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                }
            ],
            "Aggregation": {
                "Schema": [
                    {
                        "Name": "Manufacturer",
                        "Type": "string",
                        "Length": 512,
                        "RenderAs": null
                    },
                    {
                        "Name": "BIOSVersion",
                        "Type": "string",
                        "Length": 512,
                        "RenderAs": null
                    },
                    {
                        "Name": "Count",
```


Use InstructionDefinitions object inside the Tachyon connector instance.

Retrieving filtered, sorted and paginated instruction definitions

```
var settings = new Search
{
    PageSize = 200,
    Start = 1,
    Filter = new ExpressionObject
    {
        Attribute = "Name",
        Operator = "LIKE",
        Value = "%bios%"
    }
};

var definitions = connector.
InstructionDefinitions.FindInstructions
(settings);
```

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

```
                        "Type": "int64",
                        "Length": 0,
                        "RenderAs": null
                    }
                ],
                "GroupBy": "Manufacturer,BIOSVersion",
                "Operations": [
                    {
                        "Name": "Count",
                        "Type": "count"
                    }
                ]
            },
            "InstructionTtlMinutes": 60,
            "ResponseTtlMinutes": 60,
            "MinimumInstructionTtlMinutes": 10,
            "MaximumInstructionTtlMinutes": 10080,
            "MinimumResponseTtlMinutes": 10,
            "MaximumResponseTtlMinutes": 10080,
            "Workflow": null,
            "ResponseTemplateId": 1,
            "OriginalFileName": null,
            "ResponseTemplateConfiguration": {
                "Name": "default",
                "TemplateConfigurations": [
                    {
                        "Id": "mainchart",
                        "Title": "Most common BIOS
manufacturers and versions",
                        "Type": "Bar",
                        "X": "BIOSVersion",
                        "Y": "Count",
                        "Z": "Manufacturer",
                        "PostProcessor":
"processingFunction",
                        "Size": 1,
                        "Row": 1
                    }
                ],
                "PostProcessors": [
                    {
                        "Name": "processingFunction",
                        "Function": "ProcessMultiSeries
('BIOSVersion', 'Count', 'Manufacturer', '5', '5', 'false')"
                    }
                ]
            },
            "Version": "3",
            "Author": "1E",
            "IsLicensed": true,
            "UploadedTimestampUtc": "2019-03-19T10:14:39.363
Z",
            "NumberOfTimesExecuted": 0,
            "Executable": null
        }
    ]
}
```


If you want to search the same way as search-as-you-type feature, you can adjust the search criteria to include Readable payload and Description columns.

Direct Consumer API call	C# code using Consumer SDK library
Making a POST request to https://my.tachyon.server/Consumer /InstructionDefinitions/Search with following payload:	

Payload sent to https://my.tachyon.server/Consumer/InstructionDefinitions/Search

```
{
    "Start": 1,
    "PageSize": 200,
    "Filter": {
        "Operator": "OR",
        "Operands": [{
            "Attribute": "Name",
            "Operator": "LIKE",
            "Value": "%bios%"
        },
        {
            "Attribute": "Description",
            "Operator": "LIKE",
            "Value": "%bios%"
        },
        {
            "Attribute": "ReadablePayload",
            "Operator": "LIKE",
            "Value": "%bios%"
        }]
    }
}
```

Returns a similar response to the one below:

Return payload

```
{
    "TotalCount": 1,
    "Items": [
        {
            "Id": 66,
            "Name": "1E-Explorer-TachyonCore-BiosDetails",
            "Description": "Returns details of BIOS
firmware.",
            "InstructionSetId": 2,
            "InstructionSetName": "Wszystko",
            "InstructionType": 0,
            "ReadablePayload": "What BIOS firmware is
installed?",
            "Parameters": null,
            "Schema": [
                {
                    "Name": "BIOSVersion",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                },
                {
                    "Name": "Caption",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                },
                {
                    "Name": "Manufacturer",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                },
                {
                    "Name": "PrimaryBIOS",
                    "Type": "bool",
                    "Length": 0,
```

Use InstructionDefinitions object inside the Tachyon connector instance.

Retrieving instruction definitions with the word "bios" in the name, description or readable payloyad

```
var settings = new Search
{
    PageSize = 200,
    Start = 1,
    Filter = new ExpressionObject
    {
        Operator = "OR",
        Operands = new
List<ExpressionObject>
        {
            new ExpressionObject
            {
                Attribute = "Name",
                Operator = "LIKE",
                Value = "%bios%"
            },
            new ExpressionObject
            {
                Attribute =
"Description",
                Operator = "LIKE",
                Value = "%bios%"
            },
            new ExpressionObject
            {
                Attribute =
"ReadablePayload",
                Operator = "LIKE",
                Value = "%bios%"
            }
        }
    }
};

var definitions = connector.
InstructionDefinitions.FindInstructions
(settings);
```

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

```
                    "RenderAs": null
                },
                {
                    "Name": "ReleaseDate",
                    "Type": "datetime",
                    "Length": 0,
                    "RenderAs": null
                },
                {
                    "Name": "SerialNumber",
                    "Type": "string",
                    "Length": 512,
                    "RenderAs": null
                }
            ],
            "Aggregation": {
                "Schema": [
                    {
                        "Name": "Manufacturer",
                        "Type": "string",
                        "Length": 512,
                        "RenderAs": null
                    },
                    {
                        "Name": "BIOSVersion",
                        "Type": "string",
                        "Length": 512,
                        "RenderAs": null
                    },
                    {
                        "Name": "Count",
                        "Type": "int64",
                        "Length": 0,
                        "RenderAs": null
                    }
                ],
                "GroupBy": "Manufacturer,BIOSVersion",
                "Operations": [
                    {
                        "Name": "Count",
                        "Type": "count"
                    }
                ]
            },
            "InstructionTtlMinutes": 60,
            "ResponseTtlMinutes": 60,
            "MinimumInstructionTtlMinutes": 10,
            "MaximumInstructionTtlMinutes": 10080,
            "MinimumResponseTtlMinutes": 10,
            "MaximumResponseTtlMinutes": 10080,
            "Workflow": null,
            "ResponseTemplateId": 1,
            "OriginalFileName": null,
            "ResponseTemplateConfiguration": {
                "Name": "default",
                "TemplateConfigurations": [
                    {
                        "Id": "mainchart",
                        "Title": "Most common BIOS
manufacturers and versions",
                        "Type": "Bar",
                        "X": "BIOSVersion",
                        "Y": "Count",
                        "Z": "Manufacturer",
                        "PostProcessor":
"processingFunction",
                        "Size": 1,
                        "Row": 1
                    }
                ],
                "PostProcessors": [
```

```
                    {
                        "Name": "processingFunction",
                        "Function": "ProcessMultiSeries
('BIOSVersion', 'Count', 'Manufacturer', '5', '5', 'false')"
                    }
                ]
            },
            "Version": "3",
            "Author": "1E",
            "IsLicensed": true,
            "UploadedTimestampUtc": "2019-03-19T10:14:39.363
Z",
            "NumberOfTimesExecuted": 0,
            "Executable": null
        }
    ]
}
```


Searching for instruction definitions when issuing a follow-up instruction

To issue an instruction as a follow up to another instruction, additional conditions need to be taken into account.

If you were to simply retrieve all definitions as in retrieving all instruction definitions, although you'd get a list of instruction definitions that you, the caller, have permission to, you'd ignore all restrictions coming from the instruction that's the parent (the instruction you're following up on). Those restrictions stem from permissions of whoever issued the instruction you're about to follow up on. You need to use another endpoint to make sure you're searching for instruction definitions in the right context.

Retrieving all instruction definitions that can be used to issue a follow-up

You can retrieve all definitions you have permission to, like with retrieving all instruction definitions, in the context of a follow-up instruction.

This example assumes you're following up on an instruction with the Id of 12. The instruction itself is irrelevant for this example which focuses on establishing what instruction definitions could be used for a follow-up.

Direct Consumer API call	C# code using Consumer SDK library
Make a GET request to https://my.tachyon.server/Consumer/Instruction Definitions/Parent/12 to retrieve all instruction definitions the caller can issue as a follow-up. The return payload will have the same structure as the payload seen when we were retrieving all the instruction definitions in retrieving all instruction definitions. If you want to only get instruction definitions of a specific type, add a query string parameter as in retrieving only instruction definitions of a specific type. For instance, if you wanted to get all Actions you'd make this GET request: https://my.tachyon.server/Consumer/Instruction Definitions/Parent/12?instructionType=action	Use InstructionDefinitions object inside the Tachyon connector instance. **Retrieving all instruction definitions that can be used to issue follow up to instruction 12** `var definitions = connector.InstructionDefinitions.GetInstructionDefinitionsAvailableAsFollowUp(12);` The definitions object will contain the same data we've seen when retrieving all the instruction definitions in retrieving all instruction definitions.

Searching for feasible follow-up instruction definitions that match specific criteria

You can also use a search-as-you-type endpoint, like the using search-as-you-type feature, when looking for instruction definitions to issue follow-up instructions.

Direct Consumer API call	C# code using Consumer SDK library

Making a GET request to https://my.tachyon.server/Consumer /InstructionDefinitions/Parent/12/Search/Ymlvcw== will return the following response:

Response received from https://my.tachyon.server/Consumer /InstructionDefinitions/Search


```
{
    "Result": 0,
    "Error": null,
    "Instructions": [
        {
            "Id": 66,
            "Name": "1E-Explorer-TachyonCore-
BiosDetails",
            "ReadableName": "What BIOS firmware
is installed?",
            "Description": "Returns details of
BIOS firmware.",
            "Category": "SomeCategory",
            "WordMatchCount": 1,
            "InstructionType": 0,
            "InstructionTtlMinutes": 60,
            "ResponseTtlMinutes": 60,
            "ParameterHints": null,
            "Aggregatable": true,
            "IsLicensed": true,
            "NumberOfTimesExecuted": 0,
            "Executable": null
        }
    ]
}
```


Use InstructionDefinitions object inside the Tachyon connector instance.

Retrieving instruction definitions matching a search term that can be used to follow-up the instruction number 12

```
var definitions = connector.InstructionDefinitions.
GetInstructionDefinitionHintsForFollowUp("bios",
12, null);
```

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

Retrieving specific instruction definition

You can also attempt to retrieve a specific definition, either by its Id or Name.

The following two examples look for the Bios instruction definition, both using its Id, which is 66 as seen in previous examples and its name which is 1E-Explorer-TachyonCore-BiosDetails.

Direct Consumer API call

Making a GET request to https://my.tachyon.server/Consumer /InstructionDefinitions/Id/66 returns the following response:

Response received from https://my.tachyon.server/Consumer /InstructionDefinitions/Id/66


```
{
    "Id": 66,
    "Name": "1E-Explorer-TachyonCore-BiosDetails",
    "Description": "Returns details of BIOS firmware.",
    "InstructionSetId": 2,
    "InstructionSetName": "Wszystko",
    "InstructionType": 0,
    "ReadablePayload": "What BIOS firmware is installed?",
    "Parameters": null,
    "Schema": [
        {
            "Name": "BIOSVersion",
            "Type": "string",
            "Length": 512,
            "RenderAs": null
        },
        {
            "Name": "Caption",
```


C# code using Consumer SDK library

Use InstructionDefinitions object inside the Tachyon connector instance.

Retrieving a specific instruction definition by its Id

```
var definition = connector.
InstructionDefinitions.
GetInstructionDefinition(66);
```

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

```
            "Type": "string",
            "Length": 512,
            "RenderAs": null
        },
        {
            "Name": "Manufacturer",
            "Type": "string",
            "Length": 512,
            "RenderAs": null
        },
        {
            "Name": "PrimaryBIOS",
            "Type": "bool",
            "Length": 0,
            "RenderAs": null
        },
        {
            "Name": "ReleaseDate",
            "Type": "datetime",
            "Length": 0,
            "RenderAs": null
        },
        {
            "Name": "SerialNumber",
            "Type": "string",
            "Length": 512,
            "RenderAs": null
        }
    ],
    "Aggregation": {
        "Schema": [
            {
                "Name": "Manufacturer",
                "Type": "string",
                "Length": 512,
                "RenderAs": null
            },
            {
                "Name": "BIOSVersion",
                "Type": "string",
                "Length": 512,
                "RenderAs": null
            },
            {
                "Name": "Count",
                "Type": "int64",
                "Length": 0,
                "RenderAs": null
            }
        ],
        "GroupBy": "Manufacturer,BIOSVersion",
        "Operations": [
            {
                "Name": "Count",
                "Type": "count"
            }
        ]
    },
    "InstructionTtlMinutes": 60,
    "ResponseTtlMinutes": 60,
    "MinimumInstructionTtlMinutes": 10,
    "MaximumInstructionTtlMinutes": 10080,
    "MinimumResponseTtlMinutes": 10,
    "MaximumResponseTtlMinutes": 10080,
    "Workflow": null,
    "ResponseTemplateId": 1,
    "OriginalFileName": null,
    "ResponseTemplateConfiguration": {
        "Name": "default",
        "TemplateConfigurations": [
            {
```

```
                "Id": "mainchart",
                "Title": "Most common BIOS manufacturers
and versions",
                "Type": "Bar",
                "X": "BIOSVersion",
                "Y": "Count",
                "Z": "Manufacturer",
                "PostProcessor": "processingFunction",
                "Size": 1,
                "Row": 1
            }
        ],
        "PostProcessors": [
            {
                "Name": "processingFunction",
                "Function": "ProcessMultiSeries
('BIOSVersion', 'Count', 'Manufacturer', '5', '5',
'false')"
            }
        ]
    },
    "Version": "3",
    "Author": "1E",
    "IsLicensed": true,
    "UploadedTimestampUtc": "2019-03-19T10:14:39.363Z",
    "NumberOfTimesExecuted": 0,
    "Executable": null
}
```


Direct Consumer API call	C# code using Consumer SDK library
Making a GET request to https:// my.tachyon.server /Consumer /InstructionDefinitions/Name/1E-Explorer-TachyonCore-BiosDetails returns the same response seen in the previous example.	Use InstructionDefinitions object inside the Tachyon connector instance. **Retrieving a specific instruction definition by its Name** `var definition = connector.InstructionDefinitions.GetInstructionDefinition("1E-Explorer-TachyonCore-BiosDetails");` The definition object will contain the same information it did in the previous example.

Exporting Instruction Definitions

You can export Instruction Definitions back into their original XML format.

>

Direct Consumer API call	C# code using Consumer SDK library

Making a GET request to https://my.tachyon.server/Consumer /InstructionDefinitions/Id/1/export gives you a binary stream with an XML document representing requested Instruction Definition.
 	
You can also export it by name by making a GET request to https://my.tachyon.server/Consumer /InstructionDefinitions/Name/1E-Explorer-Examples-CountEventLogErrorsInLastX/export
	Use InstructionDefinitions object inside the Tachyon connector instance:
	
	Exporting specific Instruction Definition by its Id
	
	`response = connector.InstructionDefinitions.ExportDefinition(1);`
	
	Or by name:
	
	Exporting specific Instruction Definition by its Name
	
	`response = connector.InstructionDefinitions.ExportDefinition("1E-Explorer-Examples-CountEventLogErrorsInLastX");`
	
	response.ReceivedObject will be a MemoryStream holding an XML file containing the requested Instruction Definition.

You can also export multiple Instruction Definitions at the same time, specifying either their Ids or Names, giving you a zip file containing one XML file for each Instruction Definition.

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

Making a POST request with following payload if you want to export Instruction Definitions by Id:
 	
Payload sent to https://my.tachyon.server/Consumer /InstructionDefinitions/Export
 	
`{ "Ids": [ 1,2 ]}`
 	
Or this payload if you want to export them by name:
 	
Payload sent to https://my.tachyon.server/Consumer /InstructionDefinitions/Export
 	
`{ "Names": [ "1E-Explorer-Examples-CountEventLogErrorsInLastX", "1E-Explorer-Examples-DeleteCertificates" ]}`
 	
Returns a byte stream with a zip file containing requested Instruction Definitions.
	Use InstructionDefinitions object inside the Tachyon connector instance.
	
	Exporting multiple Instruction Definitions
	
	`response = connector.InstructionDefinitions.ExportDefinitions(new List<int> {1, 2});`
	
	response.ReceivedObject will be a MemoryStream containing a zip file containing one XML file for each Instruction Definition. The Consumer SDK does not support exporting multiple Instruction Definitions by name.