ITA_User Instruction Manual

RestAPI

— Version 1.7 —

Disclaimer

Trademark

※「Exastro IT Automation」is written as「ITA」in this document.

1 Table of contents

2 Introduction

This document explains the function and the operation method of REST API in ITA system.

1 ITA system Overview of REST API

This chapter explains the standard REST API for operating ITA.

1.1 About REST API

ITA provides REST API that can perform various operations to resources that are managed in ITA from external programs.

- Standard RESTAPI can be used in the menu other than the menu described in the following table. Please refer to “2 Standard REST function” for the details of standard REST API.

Table 1-1 List of Individual REST API

Menu group	Menu name	Menu ID	Reference
Symphony	Symphony class editor	2100000306	4 Using Symphony
	Symphony execution	2100000308	
	Symphony execution checking	2100000309	
Conductor	Conductor execution	2100180004	5 Conductor
	Conductor execution checking	2100180005	
Export/Import	Export menu	2100000211	3 Menu export / import
	Import menu	2100000212	
Ansible-Legacy	Execution	2100020111	6 Movement
	Check operation status	2100020112	
Ansible-Pioneer	Execution	2100020211	
	Check operation status	2100020212	
Ansible-LegacyRole	Execution	2100020312	
	Check operation status	2100020313	
Terraform	Execution	2100080009	
	Check operation status	2100080010	

2 Standard REST function

Using REST API from external programs to operate the resources managed in ITA is possible.
The following shows the calling convention.

2.1 Format of request

ITA REST API sends HTTP request to the path of each menu on ITA.

Path

https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no=(Menu ID of each menu)

e.g.) In the case of “Management console” – “system settings” menu (Menu ID:2100000202)

https:// exastro-it-automation:443/default/menu/07_rest_api_ver1.php?no=2100000202

※<HostName>: The host name “exastro-it-automation“ when installing ITA with ITA installer.

HTTP Header:

The items in the following table can be used.

Table 2-1 HTTP header parameter list

HTTP Header	Description
Host	Specify the host name of ITA RestAPI server or the IP address and port separated with colon (:)
Content-Type	Specify “application/json” It’s optional if the Method is GET.
Authorization	When accessing the menu that needs ITA authentication, specify the value of “Login ID” and “Password”* connected with half-width colon (:) then encoded with base64. It’s optional if the Method is GET.
X-Command	Can be set only when Method is POST. One of【INFO】、【FILTER】、【EDIT】can be set.

Whether the HTTP request is in uppercase or lowercase does not matter.

* If the ITA password has expired, RestAPI will turn out to be error.
Please perform request after changing the password from the login screen of Web system.
However, please follow the authentication information managed in ActiveDirectory when using the ActiveDirectory association function. (This limitation doesn’t apply to the Non-association target user of ActiveDirectory association function.)

Please refer to “User Instruction Manual_Management console” – “Usage of AcitveDirectory association function” for the details of ActiveDirectory association function.

Example of HTTP header：

In the case that Login ID is [test_loginid] and password is [test_password]

Encrypt test_loginid: test_password with base64 encoding

→[qTlmqS9fo2qcozyxBaEyp3EspTSmp3qipzD=]）

Host:<HostName>:<Port>

Content-Type:application/json

Authorization: qTlmqS9fo2qcozyxBaEyp3EspTSmp3qipzD=

X-Command: INFO

2.2 Available Methods and Commands

As a general rule, we ask you to specify POST for the HTTP request method.
However, if the user is going to access a menu that has been set to require no authentication by ITA, GET can be specified as an exception.

Method
- ├GET - ①
- │
- └POST

(X-Command) │
- ├【INFO】 - ②
- ├【FILTER】 - ③
- └【EDIT】 - ④

(1) GET (Method)

Column information (column number and column name) and the record line count and record content in normal state (discarded or active) will be returned.

・HTTP Header

None

・content parameter

None

・Response

1) Record line count

(JSON format)

The numeric value is stored key{resultdata} -> key{CONTENTS} -> key{RECORD_LENGTH}

2) Column information (Column number and column name)

(JSON format)

Stored as array with numeric key value starting from 0 in key{resultdata} -> key{CONTENTS} -> key{BODY} -> key{0}

Table 2-2 Response parameter list (Column information)

Column number	Column name
0	First column
1	Second column
⋮	⋮

3) Record information

(JSON format)(One array per row (column number and column-specific data))

Stored as array with numeric key value starting from 0 in key{resultdata} -> key{CONTENTS} -> key{BODY} -> key{(Numeric value starts from 1 to the maximum line count of the according record)}

Table 2-3 List of Response parameter (Record information)

Column number	Column data
0	First data array
1	Second data array
⋮	⋮

The table of data reponsed by Method:GET and the Json hierarchy structure is as below

Table 2-4 List of returned data

	0	1	2
0	A	B	C
1	D	E	F
2	G	H	I
3	J	K	L

▽JSON format

```
{
    "resultdata": {
        "CONTENTS": {
            " ├RECORD_LENGTH": 3,
            " └BODY": {
                "0": [
                    "A",
                    "B",
                    "C"
                ],
                "1": [
                    "D",
                    "E",
                    "F"
                ],
                "2": [
                    "G",
                    "H",
                    "I"
                ],
                "3": [
                    "J",
                    "K",
                    "L"
                ]
            }
        }
    }
}
```

(2) INFO(X-Command)

Obtain column information only.

It is possible to obtain required information only by executing X-Command (FILTER) or X-Command (EDIT).

・HTTP Header

Table 2-5 HTTP Header parameter list

HTTP header	Value
X-Command	INFO

・content parameter

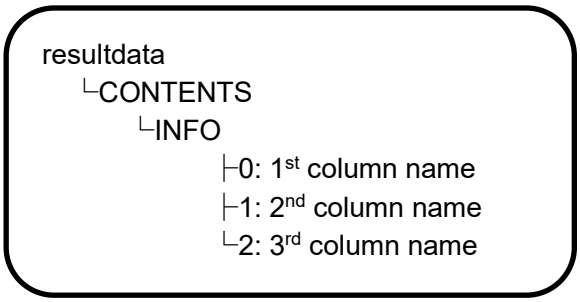
None

・Response

1) Column information (Column number and column name)

(JSON format)

Stored as array with numeric key value starting from 0 in key{resultdata} -> key{CONTENTS} -> key{INFO}



(3) FILTER(X-Command)

The Column information (Column number and column name) and the line count and record content of the whole record in normal status (discarded or active) that meets the criteria specified in parameter is returned.

Table 2-6 HTTP header parameter list

HTTP header	Value
X-Command	FILTER

・content parameter

1) Filter format

By follow the format of the column that can be filtered by the display filter on the Web console, the following types of filters can be specified for each column.

・NORMAL - Normal LIKE search

・RANGE - Range search from level 1~5

Also, if the column is displayed in pulldown menu in the display filter on web console, specifying LIST (“OR” search with multiple exact match conditions. Specify multiple conditions in array) is available.

2) Specification format

Specify in JSON format. Store the filter criteria according to the format of filter type. When turning the data into JSON format, specify associative arrays for each column nested in one associative array. If multiple columns are stored in separated associative arrays, it means they are connected with AND relationship.

In addition, please set the format and criteria of filter criteria and store it in an associate array then put it in the associative array of each column. If multiple associative array of filter conditions is stored in single associative array, it means that they are connected with OR relationship.

・Parameter specification example

e.g.) Describing FILTER parameter

Column 2 is "ID" (primary key column), Column 4 is the content of "Remarks".
In the case of extracting the record whose "number" is 5 or more, and having "ABC" in the "Remarks"

↓Extraction image

	Column1	Column2	Column	Column
row	Column1	ID	Coumn3	Remarks
1	*****	1	*****	ABCDE
2	*****	2	*****	GHIJK
3	*****	3	*****	ABCDE
4	*****	4	*****	GHIJK
5	*****	5	*****	GHIJK
6	*****	6	*****	ABCDE
⋮	⋮	⋮	⋮	⋮

▽JSON format

```
{
    "2": {
        "RANGE": {
            "START": 5
        }
    },
    "4": {
        '"NORMAL": "ABC"
    }
}
```

e.g.) Describing FILTER parameter (2)

Column 2 is "ID" (primary key column).
In the case of extracting the record whose "number" is from 10~99 or is 1,2,or 5.

↓Extraction image

	Column1	Column2	…
Row	Column1	ID	…
1	*****	1	…
2	*****	2	…
3	*****	3	…
4	*****	4	…
5	*****	5	…
⋮	⋮	⋮	…
10	*****	10	…
⋮	⋮	⋮	…
99	*****	99	…
⋮	⋮	⋮	…

▽JSON format

```
{
    "2": {
        "RANGE": {
            "START": "10",
            "END": "99"
        },
        "LIST": [
            "1",
            "2",
            "5"
        ]
    }
}
```

e.g.) Describing FILTER parameter (2)

Column 2 is "ID" (primary key column), Column 5 is "last update date"(date type/date time type).
In the case of extracting the record whose "ID" is in 1 to 100 and the last update time is in 2016/8/1(00:00:00) to 2016/12/31(23:59:59)

▽JSON format

```
{
    "2": {
        "RANGE": {
            "START": "1",
            "END": "100"
        }
    },
    "5": {
        "RANGE": {
            "START": "2016/08/01 00:00:00",
            "END": "2016/12/31 23:59:59"
```

```
        }
    }
}
```

・reponse

1) Record line count
(JSON format)
The numeric value is stored key{resultdata} -> key{CONTENTS} -> key{RECORD_LENGTH}

2) Column information (Column number and column name)
(JSON format)
Stored as array with numeric key value starting from 0 in key{resultdata} -> key{CONTENTS} -> key{BODY} -> key{0}

Table 2-7 Response parameter list (Column information)

Column number	Column name
0	First column
1	Second column
⋮	⋮

3) Record information
(JSON format) (One array per row (column number and column-specific data))
Stored as array with numeric key value starting from 0 in key{resultdata} -> key{CONTENTS} -> key{BODY} -> key{(Numeric value starts from 1 to the maximum line count of the according record)}

Table 2-8 List of Response parameter (Record information)

Column number	Column data	Description
0	First data array	
1	Second data array	
⋮	⋮	

※ The Json hierarchy is same as Method:GET.

For details on how to search for Access permission roles, please refer to “Exastro-ITA_User_instruction_manual_Role-based_access_control_for_data_records".

(4) EDIT(X-Command)
Register record or update/discard/restore existing record.

・HTTP Header

Table 2-9 HTTP header parameter list

HTTP header	Value
X-Command	EDIT

・Parameter

1) Specification format.
Please specify in JSON format.

Please specify one record to one array based on the column information obtained from INFO and store the specified arrays as the element of an array encoded in JSON type then send the array as the context of HTTP request.
For the column number 0, “execution process type”, please specify one of “Register”, “Update”, “Discard”, “Update”.

Example(1) Register

Column 0 is “execution process type”, column 1 is “Discard” , column 2 is “ID” (primary key column)
~ omitted ~
Column 10 is “Remarks, column 11 is "Last update date/time", column 12 is "Last update date/time for updating", and column 13 is "Last updated by".
In the case of adding 2 records.

▽JSON format

```
[
    {
        "0": {
            "0": " Update ",
            "1": "",
            "2": "",
                ・・・（Omitted）・・・
            "10": "Remarks",
            "11": "",
            "12": "",
            "13": ""
        },
        "1": {
            "0": " Update ",
            "1": "",
            "2": "",
                ・・・（Omitted）・・・
            "10": "Remarks",
            "11": "",
            "12": "",
            "13": ""
        }
    }
]
```

Example(2) Update

Column 0 is “execution process type”, column 1 is “Discard” , column 2 is “ID” (primary key column)
~ omitted ~
Column 9 is “Remarks, column 10 is "Last update date/time", column 11 is "Last update date/time for updating", and column 12 is "Last updated by".

In the case of updating the record with “ID” 10.

▽JSON format

```
[
    {
        "0": "Update",
        "1": "",
        "2": "10",
        "9": "Remarks",
        "10": "2016/08/01 12:30:45",
        "11": "Last update date/time for updating",※
        "12": "Administrator"
    }
]
```

※Method： Please set the "Last update date/time for updating" obtained from GET、X-Command：FILTER. This data is used to avoid old data to overwrite update new data.
"Last update date/time for updating" begins from “T_”.

Example (3)　Register (With file upload)

Column 0 is “execution process type”, column 1 is “Discard” , column 2 is “ID” (primary key column)
~ omitted ~
Column 5 is “Remarks, column 6 is "Last update date/time", column 7 is "Last update date/time for updating", and column 8 is "Last updated by".

▽JSON format

・ In the case of adding 1 record with file upload.

```
{
    "0": {
        "0": "Register",
        "3": "PV05004",
        "4": "20191226095004.yml",
        "5": "TEST"
    }
    "UPLOAD_FILE": [
        {
            "4": "<The value of target file encoded by base64>"
        }
    ]
}
```

・ In the case of adding 2 records with file upload.

```
{
    "0": {
        "0": "Register ",
        "3": "PV05004",
```

```
            "4": "20191226095004.yml",
            "5": "TEST"
        },
        "1": {
            "0": "Register ",
            "3": "PV15004",
            "4": "20191226095004.yml",
            "5": "TEST"
        },
        "UPLOAD_FILE": [
            {
                "4": "<The value of target file encoded by base64>"
            },
            {
                "4": "<The value of target file encoded by base64>"
            }
        ]
}
```

※For UPLOAD_FILE, specify the value of target file encoded by base64 to perform file upload.
※When uploading files, please add the files in the order of element to “UPLOAD_FILE”.
※If you want to change the size limit of file uploads, you need to change the PHP settings. Refer to section “8.1 Troubleshooting” for information on what items to change.
Refer to “Reference” configuration settings when installing” for more information on default values (different document).

・Responce

1) Execution result of each record

(JSON format)

Stored as array in key{resultdata} -> key{LIST} -> key{NORMAL} -> key{register、update、delete、error}

Table 2-10 Key parameter list

key	Value type	
name	String	The name of operation result type
ct	Numeric	Record count of each operation result

2) Operation result of each record

(JSON format)

Stored as array with numeric key value starting from 0 in key{resultdata}->key{LIST}->key{RAW}->key{Record number sent as parameter (No need to send column information), number starts from 0 as default}

Table 2-11 Key parameter list

key	Value type	
0	String	Result code (refer to the table below)
1	String	Detail code (refer to the table below)

2	String	Error message

・Response hierarchy

```
resultdata
  └LIST
      ├NORMAL
      |  ├register: {name:,ct:}
      |  ├update:  {name:,ct:}
      |  ├delete:   {name:,ct:}
      |  └error:    {name:,ct:}
      |
      └RAW
         ├0:{0:,1:,2:}
         ├1: {0:,1:,2:}
         ├2: {0:,1:,2:}
         ├ ・
         ├ ・
           ・
```

・Response

The returned response is stored in JSON format

```
{
    "status": "SUCCEED",
    "resultdata": {
        "LIST": {
            "NORMAL": {
                "register": {
                    "name": "Register",
                    "ct":
                },
                "update": {
                    "name": “Update",
                    "ct":
                },
                "delete": {
                    "name": "Discard",
                    "ct":
                },
                "revive": {
                    "name": "Restore",
                    "ct":
                },
                "error": {
                    "name": "Error",
                    "ct":
                }
            },
```

```
                "RAW": [
                    [
                        ・・・Appendix: Result code/Detail code list・・・
                    ],
                        ・・・(omitted)・・・
                ]
            }
        }
}
```

Appendix: Result code/Detail code list

Operation type	Result code	Detail code	Description
Register	000	201	Successfully registered.
Register	002	000	Required item not entered.
Register	002	000	Some items are duplicated with the record.
Register	002	000	There is a record that violates duplication prohibition.
Register	002	000	The length of the input value exceeds the specified number of bytes.
Register	002	000	The input value [value including NULL byte characters etc] is invalid.
Register	002	000	A value other than half-width integer was entered.
Register	002	000	Value out of range.
Register	002	000	The entered value is below the minimum value or above the maximum value.
Register	002	000	Input condition is not satisfied.
Register	002	000	A non-numeric value was entered.
Register	002	000	Tab or line feed was entered.
Register	002	000	Tab was entered.
Register	002	000	Input value is out of range.
Register	002	000	Input value is out of the range that can be processed normally by PHP function (checkdate).
Register	002	000	An unavailable value was selected.
Register	002	000	An item (primary key) that cannot be specified during registration was specified.
Register	-	-	Do not have maintenance permission.
Update	000	200	Successfully Updated.
Update	002	000	Required fields are not entered.
Update	002	000	Some items are duplicated with the record.
Update	002	000	There is a record that violates duplication prohibition.
Update	002	000	The length of the input value exceeds the specified number of bytes.
Update	002	000	The input value [value including NULL byte characters] is invalid.
Update	002	000	Value other than a half-width integer has been entered.
Update	002	000	Value out of range.
Update	002	000	The entered value is below the minimum value or above the maximum value.
Update	002	000	The input conditions do not met.
Update	002	000	A non-numeric value was entered.
Update	002	000	Tab and line feed was entered.
Update	002	000	Tab was entered.
Update	002	000	Input value was out of range.
Update	002	000	Input value was out of the range that can be processed normally by PHP function

			(checkdate).
Update	002	000	An unavailable value was selected.
Update	003	000	The update execution stop due to update of record from another session.
Update	003	000	An update to a discarded record was about to be performed.
Update	101	000	The column of update target was not specified.
Update	-	-	Do not have maintenance permission.
Discard	000	210	Successfully discarded.
Discard	002	000	The length of the input value exceeds the specified number of bytes.
Discard	002	000	The input value [value including NULL byte characters] is invalid.
Operation type	Result code	Detail code	Description
Discard	002	000	The entered value was below the minimum value or above the maximum value.
Discard	002	000	The input condition was not satisfied.
Discard	002	000	Tab was entered.
Discard	003	000	The discard execution stopped due to update of record from another session.
Discard	003	000	Discarding a discarded record was about to be performed.
Discard	101	000	The column of discard target was not specified.
Discard	-	-	Do not have maintenance permission
Restore	000	200	Successfully restored
Restore	002	000	Required item not entered
Restore	002	000	An item that cannot be updated was about to be updated when performing restoration.
Restore	002	000	Some items are duplicated to the record.
Restore	002	000	There is a record that violates duplication prohibition.
Restore	002	000	The length of the input value exceeds the specified number of bytes.
Restore	002	000	The input value [value including NULL byte characters] was invalid.
Restore	002	000	The entered value was below the minimum value or above the maximum value.
Restore	002	000	The input condition was not satisfied.
Restore	002	000	Tab was entered.
Restore	003	000	The restore execution stopped due to restore of record from another session.
Restore	003	000	Restoring restored record was about to be performed.
Restore	101	000	The column of restore target can't be identified.
Restore	-	-	Do not have maintenance permission.
Display	-	-	Validation error
Display	-	-	One of the following ("All records", "Exclude discarded records" and "Only discarded records") is not selected
-	000	000	Skip the operation and go on to the next record.

3 Menu export / import

3.1 RestAPI for menu export

It is possible to perform menu export with RestAPI.
The available function is same as the operation in “Export menu” menu in “Export/Import” menu group.

Table 3-1 Menu list

Menu group	Menu name	Menu ID
Export/Import	Export menu	2100000211

3.1.1 Request format

Send HTTP request with the following information.

・Path
https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no=MenuID
Please refer to “**Table 3-3 List of parameters that can be specified for X-Command**” for Menu ID.

・HTTP Header

Table 3-2 HTTP header parameter list

HTTP Header	Description
Content-Type	Specify “application/json”
Authorization	When accessing the menu that needs ITA authentication, specify the value of “Login ID” and “Password”* connected with half-width colon (:) then encoded with base64.
X-Command	EXECUTE INFO These two can be chosen

Parameters that can be specified for X-Command

Table 3-3 List of parameters that can be specified for X-Command

X-Command	Description	Target menu	Menu ID
INFO	Obtain the list of menu that can be exported	Export menu	2100000211
EXECUTE	Execute menu export	Export menu	2100000211

The following is the explanation of each X-command parameter.

3.1.2 **INFO**

Output the list of the menu that can be exported.

・Parameter

No parameter to specify.

・Response

The returned response is stored in JSON format.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "MENU_LIST": {
            "Menu group ID ": {
                "menu_group_name": "Menu group name",
                "menu": [
                    {
                        "menu_id": "Menu ID ",
                        "menu_name": "Menu name"
                    },
                    {
                        ・・・(omitted)・・・
                    }
                ]
            },
            " Menu group ID ": {
                        ・・・(omitted)・・・
            }

        }
    }
}
```

Table 3-4 Response item list

Item name	Remarks
Menu group ID	An array with menu group ID as key and consisted of menu.
menu_group_name	Menu group name
menu_id	Menu ID
menu_name	Menu name

3.1.3 **EXECUTE**

Specify target menu and execute menu export.

・Parameter

Please specify the following to “content” in JSON format.

Table 3-5　Export menu parameter

Parameter name	Value
Menu group ID	Menu ID
Dp_mode	Mode 1.Override 2.Add
Abolished_type	Abolition data 1.Normal 2.Without disuse data
Specified_timestamp	Time specification Only input something when Mode no.2, Time specification is chosen. The date format is as following: YYYY-MM-DD H:i E.g) 2020-01-01 00:00

※ Menu group ID and Menu ID is the value in the return value of INFO.

Example) JSON description

```
{
    "2100000002": [
        2100000202,
            ・・・(omitted)・・・
        2100000222
    ],
    "2100000003": [
            ・・・(omitted)・・・
    ],
    “dp_mode”:”1”,
    “abolished_type”:”1”
    “specified_timestamp”:” 2020-01-01 00:00”
}
```

・Response

The returned response is stored in JSON format. Please refer to the following for the response items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "TASK_ID": "Execution No ",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
```

```
    }
}
```

Table 3-6 Response item list

Item name	Remarks
TASK_ID	Operation No. Users can check the execution status by searching the Operation No. in “Export/Import menu list”.
RESULTCODE	The code of execution status 000：Normal end 002：Not able to perform execution
RESULTINFO	Detailed information

3.2 RestAPI for Import menu

It is possible to import menus with RestAPI.
The functions available are the same as the operation in “Import menu” menu in “Export/Import” menu group.

Table 3-7 menu list

Menu group	Menu name	Menu ID
Export/Import	Import menu	2100000212

3.2.1 Request format

Send HTTP request with the following information.

・Path

https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no=MenuID

Please refer to “**Table 3-9 List of parameters that can be specified for X-Command**” for Menu ID.

・HTTP Header

Table 3-8 HTTP header parameter list

HTTP Header	Description
Content-Type	Specify “application/json”
Authorization	When accessing the menu that needs ITA authentication, specify the value of “Login ID” and “Password”* connected with half-width colon (:) then encoded with base64.
X-Command	EXECUTE INFO These two can be chosen

Parameters that can be specified for X-Command

Table 3-9 List of parameters that can be specified for X-Command

X-Command	Description	Target menu	Menu ID
UPLOAD	Upload the exported kym file and output the list of menu that can be imported.	Import menu	2100000212
EXECUTE	Select the import target menu and execute import.	Import menu	2100000212

The following is the explanation of each X-command parameter.

3.2.2 **UPLOAD**

Upload the exported file.
Send the file encoded with base64 as parameter.

・parameter
Please specify the following in JSON format to “content.

Table 3-10 Import menu UPLOAD parameter list

Parameter name	Value
name	Target file name
base64	Specify the value of target file encoded in base64.

1）UPLOAD Json description example

```
{
  "zipfile":{
    "name":"ita_exportdata_20191224092830.kym",
    "base64":"・・・(omitted)・・・"
  }
}
```

・Response
The returned response is stored in JSON format.

```
{
    "status": ""SUCCEED",
    "resultdata": {
        "upload_id": "Upload ID ",※
        "data_portability_upload_file_name": "File name",
        “dp_mode”:”1”,
        “abolished_type”:”1”,
        "IMPORT_LIST": {
            "Menu group ID ": {
                "menu_group_name": "Menu group name,
                "menu": [
                    {
                        "menu_id": "Menu ID ",
                        "menu_name": "Menu name"
                    }
                      ・・・(omitted)・・・

                ],
            " Menu group ID ": {
                      ・・・(omitted)・・・
                ],

                "RESULTCODE": "Result code",
                "RESULTINFO": "Detailed information"
```

```
            }
        }
    }
}
```

※ “upload_id” is used when executing import(EXECUTE)

Table 3-11 Response item list

Item name	Remarks
upload_id	The value given when upload succeeded. Used in EXECUTE.
data_portability_upload_file_name	File name
Dp_mode	Mode 1:Override 2:Add
Abolished_type	Abolition information 1:Normal 2:Without disuse data
Specified_timestamp	
Menu group ID	An array with menu group ID as key and consisted of menu.
menu_group_name	Menu group name
menu_id	Menu ID
menu_name	Menu name
RESULTCODE	The code of execution status 000：Normal end 002：Not able to perform execution
RESULTINFO	Detailed information

3.2.3 **EXECUTE**

Perform import based on the uploaded file.
Users can specify the target menu group, menu ID, import execution mode.

・Parameter

Please specify the following items in JSON format to “content”.

Table 3-12 Import menu EXECUTE parameter

Parameter name	Value	Remarks
Menu group ID	Menu ID	
upload_id		Add prefix “A_” to the value obtained from the return value of UPLOAD.
data_portability_upload_file_name	File name	

1）EXECUTE Json description example
{ "2100070001": [

```
            2100070001,
            2100070002,
            2100070003
        ],
        "2100020002": [
            ・・・(omitted)・・・
        ],
        "upload_id": "A_2019121709033577204023 9",※
        "data_portability_upload_file_name": "ita_exportdata_20191213095733.kym"
}
```

※Use the “upload_id” obtained from UPLOAD with “A_” added in the front of the value.

・Response

The returned response is stored in JSON format. Please refer to the following for the response items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "TASK_ID": "Execution No. of menu import",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
    }
}
```

Table 3-13 Response item list

Item name	Remarks
TASK_ID	Operation No Users can check the execution status by searching the Operation No in “Export/Import menu list”.
RESULTCODE	The code of execution status 000：Normal end 002：Not able to perform execution
RESULTINFO	Detailed information

4 Using Symphony

4.1 RestAPI for registering operations for Symphony.

It is possible to use Symphony with RestAPI.
The functions available are the ones that corresponds to the Symphony Menu group's "Symphony Class edit" menu.

Table 4-1 Menu list

Menu group	Menu name	Menu ID
Symphony	Symphony class edit	2100000306

4.1.1 Request type

Send HTTP request with the following information.

・Path
https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no= MenuID
Please refer to "Table 4-3 List of parameters that can be specified for X-Command" for Menu ID.

・HTTP header

Table 4-2 HTTP header parameter list

HTTP Header	Description
Content-Type	Specify "application/json"
Authorization	When accessing the menu that needs ITA authentication, specify the value of "Login ID" and "Password"* connected with half-width colon (:) then encoded with base64.
X-Command	INFO FILTER EDIT These three can be chosen

Parameters that can be specified for X-Command

Table 4-3 List of parameters that can be specified for X-Command

X-Command	Description	Target menu	Menu ID
INFO	Obtains the Symphony class column information	Symphony class edit	2100000306
FILTER	Refers the records that matches the Symphony class parameters.	Symphony class edit	2100000306
EDIT	Registers the Symphony class.	Symphony class edit	2100000306

The following is the explanation of each X-command parameter.

4.1.2 **INFO**

Obtains the Symphony class column information
※For more details, please refer to Chapter 2, Standard REST Function – “INFO(X-Command)

4.1.3 **FILTER**

Obtains the column information (column number and name) of the records that match the condition specified in the Parameter, as well as the number of rows, the contents of the records, and the column information of all records with normal status(obsolete or active).
※For more details, please refer to Chapter 2, Standard REST Function – “FILTER(X-Command)

4.1.4 **EDIT**

Registers, Edits, abolishes or revives Symphony classes.

・HTTP header

Table 4-4 HTTP header parameter list

HTTP header	Value
Method	POST
X-Command	EDIT

・Parameter

1) Specify format
For information regarding the parameter specification items for each execution type, refer to the following Parameter Specification Items.
When specifying "Update”,"Abolish", or "Revive" for Item number 7,
Please set the “Last modified date for update “obtained from the X-Command:FILTER.
This data prevents overtaking updates.
The “Last modified date for update” starts with a “T”.

Table 4-5 Symphony class parameter list

Item No.	Parameter name	Remarks
0	Process type	Register, Edit, Abolish, Revive.
2	Symphony Class ID	Blank when registering
3	Symphony name	
4	Description	
5	Remarks	
7	Last modified date for update	T_XXXXXXXXXXXXXXXXXXX
9	Movement details	Movement details See the table below for more information (Table 4-6)

Table 4-6 Movement details list

Item no.	Parameter name	Remarks
0	Orchestrator ID	Orchestrator ID Please see the table below (4-8)
1	Movement ID	Movement ID Please see the “Movement list” menu.
2	Temporary stop	OFF:Blank ON:checkedValue
3	Description	
4	Operation ID(Individually specified)	Operation ID Please see the “Input operation” menu.

Table 4-7 Symphony Class parameter Movement details

Parameter specified items (Register/Update)

```
"9": [
    {
        "0": "Orchestrator ID",
        "1": "Movement ID",
        "2": "Temporary stop(OFF:/ON:checkedValue)",
        "3": "Description",
        "4": "Operation(Individually specified)"
    },
    {
    ///Add more if you want to run multiple Movements///
    }

]
```

Table 4-8 Orchestrator ID Table

ID	Status
3	Ansible Legacy
4	Ansible Pioneer
5	Ansible Legacy Role
10	Terraform

Parameter specified items (Register/Update)

```
{
    "0": "Process type：<Register or Update>",
    "2": "Symphony class ID ",
    "3": "Symphony name",
    "4": "Description",
    "5": "Remarks",
    "7": " Last modified date for update ",
    "9": [
        {
```

```
                "0": "Orchestrator ID",
                "1": "Movement ID",
                "2": "Temporary stop (OFF:/ON:checkedValue)",
                "3": "Description",
                "4": "Operation (Individually specified)"
                }
        ///Add more if you want to run multiple Movements///
    ]
}
```

Parameter specified items (Abolish/Revive)

```
{
    "0": "Process type：<Abolish　or Revive>",
    "2": "Symphony class ID",
    "7": " Last modified date for update "
}
```

※Please set the “Last modified date for update “obtained from the X-Command: FILTER.
This data prevents overtaking updates.
The “Last modified date for update” starts with a “T”.

Example) JSON Description: When running multiple process types.

```
[
    {
        "0": "Register",
        "2": "",
        "3": "DEMO_001_20191224135448_0",
        "4": "demo_001_20191224135448_0",
        "7": "",
        "9": [
            {
                "1": 3,
                "2": 1,
                "3": "checkedValue",
                "4": "DEMO_MOVE_0",
                "5": 1
            },
            {
                "1": 3,
                "2": 2,
                "3": "",
                "4": "DEMO_MOVE_1",
                "5": ""
            }
        ]
    },
```

```
        {
            "0": "Update",
            "2": 1,
            "3": "DEMO_001_20191224135448_1",
            "4": "demo_001_20191224135448_1",
            "7": "T_20191224113132971799",
            "9": [
                {
                    "1": 3,
                    "2": 1,
                    "3": "",
                    "4": "DEMO_MOVE_0",
                    "5": 1
                }
            ]
        },
        {
            "0": "Abolish",
            "2": 2,
            "7": "T_20191224135437197447"
        },
        {
            "0": "Revive",
            "2": 4,
            "7": "T_20191224135449793941"
        }
]
```


・Response

For more information regarding the processing results of each of the records, please see XXXXXX

4.2 RestAPI for Symphony execution

Operating RestAPI from Symphony is possible.
The available function is same as the operation in “Symphony execution” and Symphony execution checking” menu in “Symphony” menu group.

Table 4-9 Target menu list

Menu group	Menu name	Menu ID
Symphony	Symphony execution	2100000308
	Symphony execution checking	2100000309

4.2.1 **Request type**

Send HTTP request with the following information.

・Path

https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no=MenuID

Please refer to “Table 5-11 List of parameters that can be specified for X-Command” for Menu ID.

・HTTP Header

Table 4-10 HTTP header parameter list

HTTP Header	Description
Method	POST only
Content-Type	Specify “application/json”
Authorization	When accessing the menu that needs ITA authentication, specify the value of “Login ID” and “Password”* connected with half-width colon (:) then encoded with base64.
X-Command	EXECUTE CANCEL SCRAM RELEASE These four can be chosen

Parameters that can be specified for X-Command

Table 4-11 List of parameters that can be specified for X-Command

X-Command	Description	Target menu	Menu ID
EXECUTE	Execute Symphony	Symphony execution	2100000308
CANCEL	Cancel scheduled Symphony execution	Symphony execution checking	2100000309
SCRAM	Perform Symphony emergency stop	Symphony execution checking	2100000309
RELEASE	Release Symphony pause point	Symphony execution checking	2100000309

The following is the explanation of each X-command parameter.

4.2.2 **Response item**

The following is the explanation of the response items in each X-command parameter.

Table 4-12 Response item list

Item name	Remarks
SYMPHONY_INSTANCE_ID	Used when operating SYMPHONY instances
MOVEMENT_SEQ_NO	Used in RELEASE only
RESULTCODE	The code of execution status 000：Normal end 001：Execution unavailable 002：Scheduled cannot be cancelled 003：Cannot perform emergency stop 004：Cannot unpause
RESULTINFO	Detailed information

4.2.3 **EXECUTE**

Specify Symphony class and Operation then perform Operation execution. It is possible to specify scheduled execution date/time and skip, Operation ID to each Movement registered in Symphony class individually.

・Parameter

Please specify the following items in JSON format to “content”.

Table 4-13 Operation ID individual specification parameter list

Parameter name	Value
SYMPHONY_CLASS_NO	Symphony class ID
OPERATION_ID	Operation ID
PRESERVE_DATETIME	Scheduled execution date(YYYY/MM/DD tt:mm)
OPTION	With/without skip, individual specification array of Operation ID

・Specify OPTION

In OPTION, it is possible to specify skip, Operation ID to each Movement individually in array format.

・Hierarchy of Movement element

```
├1 (Execution order of Movement)
|       ├SKIP   -   YES or NO
|       └OPERATION_ID   -   (Operation ID specified individually)
├2 (Execution order of Movement)
|       ├SKIP   -   YES or NO
|       └OPERATION_ID   -   (Operation ID specified individually)
・
・
```

1）EXECUTE　Json description example

Symphony class ID is 1, Operation ID is 1001, and scheduled date and time is 2016/01/01 00:00, In addition, skip the first executed Movement ,and specified the operation ID of the second executed Movement to 2001

▽Description in Json format

```
{
    "SYMPHONY_CLASS_NO": 1,
    "OPERATION_ID": 1001,
    "PRESERVE_DATETIME": "2016/01/0100:00",
    "OPTION": {
        "1": {
            "SKIP": "YES"
        },
        "2": {
            "OPERATION_ID": 2001
        }
    }
}
```

Figure 6.1-1 EXECUTE　Json description example

・Response

The returned response is stored in JSON format

```
{
    "status": "Successfully executed or not",
    "resultdata": {
        "SYMPHONY_INSTANCE_ID": "Execution No ",※
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
    }
}
```

※Used to operate (INFO, CANCEL, SCRAM, RELEASE) the instance after execution.

4.2.4 **CANCEL**

Specify the instance ID of the registered Symphony whose execution date is scheduled and cancel the schedule.

・Parameter

Please specify the following items in JSON format to “content”.

Table 4-14　Symphony execution schedule cancellation parameter list

Parameter name	Value
SYMPHONY_INSTANCE_ID	Symphony instance ID※

※The value obtained from the return value of EXECUTE.

・Response

The returned response is stored in JSON format. Please refer to the following for the returned items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        " SYMPHONY_INSTANCE_ID": " Symphony instance ID during execution",
        "RESULTCODE": "Result",
        "RESULTINFO": "Detailed information"
    }
}
```

4.2.5 **SCRAM**

Specify the instance ID of the executing Symphony and perform emergency stop.

・Parameter

Please specify the following items in JSON format to “content”.

Table 4-15 Symphony execution emergency stop parameter list

Parameter name	Value
SYMPHONY_INSTANCE_ID	Symphony instance ID※

※The value obtained from the return value of EXECUTE.

・Response

The returned response is stored in JSON format. Please refer to the following for the returned items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        " SYMPHONY_INSTANCE_ID": "Symphony instance ID during execution",
        "RESULTCODE": "Result",
        "RESULTINFO": "Detailed information"
    }
}
```

4.2.6 **RELEASE**

Specify Symphony instance ID and Movement order then release the points that “pause” are set.

・Parameter

Please specify the following items in JSON format to “content”.

Table 4-16 Symphony execution unpause parameter list

Parameter name	Value
SYMPHONY_INSTANCE_ID	Symphony instance ID※
MOVEMENT_SEQ_NO	The sequence number of Movement

※The value obtained from the return value of EXECUTE.

・Response

The returned response is stored in JSON format. Please refer to the following for the returned items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "SYMPHONY_INSTANCE_ID": "Symphony instance ID during execution ",
        "MOVEMENT_SEQ_NO": "The sequence number of the executed Movement in Symphony class",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
    }
}
```

4.3 RestAPI for Symphony execution checking

Operating RestAPI from Symphony is possible.
The available function is same as the operation in “Symphony execution” and Symphony execution checking” menu in “Symphony” menu group.

Table 5-17 Target menu list

Menu group	Menu name	Menu ID
Symphony	Symphony execution checking	2100000309

4.3.1 **Request type**

Send HTTP request with the following information.

・Path

https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no= MenuID

Please refer to “Table 5-19 List of parameters that can be specified for X-Command” for Menu ID.

・HTTP header

Table 4-18 HTTP header parameter list

HTTP Header	Description
Content-Type	Specify “application/json”
Authorization	When accessing the menu that needs ITA authentication, specify the value of “Login ID” and “Password”* connected with half-width colon (:) then encoded with base64.
X-Command	INFO Can be specified.

Parameters that can be specified for X-Command

Table 4-19　List of parameters that can be specified for X-Command

X-Command	Description	Target menu	Menu ID
INFO	Check the execution status of Symphony and return the status.	Symphony execution checking	2100000309

The following is the explanation of each X-command parameter.

4.3.2 **Response item**

The following is the explanation of the response items in each X-command parameter.

Table 4-20　Response item list

Item name	Remarks
SYMPHONY_INSTANCE_ID	Used when operating SYMPHONY instances
RESULTCODE	The code of execution status 000：Normal end
RESULTINFO	Detailed information

4.3.3 **INFO**

Specify the instance ID of Symphony during execution and obtain the execution information.

・Parameter

Please specify the following items in JSON format to “content”.

Table 4-21　Symphony execution obtain information parameter list

Parameter name	Value
SYMPHONY_INSTANCE_ID	Symphony instance ID※

※The value obtained from the return value of EXECUTE.

・Response

The returned response is stored in JSON format.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "SYMPHONY_CLASS_ID": "1",
        "SYMPHONY_INSTANCE_INFO": {
            "SYMPHONY_INSTANCE_ID": 1,
                ・・・(omitted) refer to the following ① for the items・・・
            "FOCUS_MOVEMENT": 1
        },
        "MOVEMENTS": [
            {
                "CLASS_ITEM": {
                    "ORCHESTRATOR_ID": "3",
                        ・・・(omitted) refer to the following ② for the items・・・
                    "NEXT_PENDING": "checkedValue"
                },
```

```
                    "INS_ITEM": {
                        "STATUS": "11",
                            ・・・(omitted) refer to the following ③ for the items・・・
                        "OPERATION_NAME": null
                    }
                }
                ・・・・
            ],
            "RESULTCODE": "000",
            "RESULTINFO": ""
        }
}
```


① The Symphony instance information array stored in SYMPHONY_INSTANCE_INFO

Table 4-22 Instance array list

Key	Content
SYMPHONY_INSTANCE_ID	Symphony instance ID
I_SYMPHONY_CLASS_NO	Class ID of the instance
I_SYMPHONY_NAME	Name of the instance
I_DESCRIPTION	Description of the instance
STATUS_ID	Execution status. Refer to Table 5-25 for details
ABORT_EXECUTE_FLAG	Emergency stop flag. Not issued: 1 Issued: 2
OPERATION_NO_UAPK	Operation NO
OPERATION_NO_IDBH	Operation ID
OPERATION_NAME	Operation name
TIME_BOOK	Scheduled execution date/time
TIME_START	Start date/time
TIME_END	End date/time
MOVEMENT_LENGTH	Number of registered Movement
FOCUS_MOVEMENT	The sequence of current execution Movement

② The Movement class information stored in CLASS_ITEM

Table 4-23 Movement class information list

Key	Content
ORCHESTRATOR_ID	Orchestrator ID. Mapping table is in below table 5-26
PATTERN_ID	Movement ID
PATTERN_NAME	Movement name
THEME_COLOR	<For Web> The color of the circle icon set on Web screen
MOVEMENT_SEQ	The sequence number in Symphony class
DESCRIPTION	The description entered in Symphony class editor screen
NEXT_PENDING	If pause is set：checkedValue

③ The Movement instance information stored in INS_ITEM

Table 4-24 Movement instance information list

Key	Content
STATUS	Execution status. Refer to Table 5-25 for details
RELEASED	Pause is set: 1 Pause is released: 2
EXECUTION_NO	Movement instance ID
JUMP	<For Web> Transit target URL
ABORT_RECEPTED	Emergency stop request 1: Not received 2: received
SKIP	If skip is set: 1
TIME_START	Start date/time
TIME_END	End date/time
OPERATION_ID	Individually specified Operation ID
OPERATION_NAME	Individually specified Operation Name

Table 4-25 Mapping table of status ID during Symphony instance execution

ID	Status
1	Unexecuted
2	Unexecuted (schedule)
3	Executing
4	Executing (delay)
5	Normal end
6	Emergency stop
7	Abend
8	Unexpected error
9	Schedule cancellation

Table 4-26 Mapping table of Orchestrator ID

ID	Status
3	Ansible Legacy
4	Ansible Pioneer
5	Ansible Legacy Role
10	Terraform

Table 4-27 Mapping table of status ID during Movement instance execution

ID	Status
1	Not executed
2	Preparing
3	Executing
4	Executing(delayed)
5	Execution completed
6	Abend
7	Emergency stop
8	Holding
9	Nomal end
10	Preparation error
11	Unexpected error
12	Skip completed
13	Holding after skip
14	Skip end

5 Conductor

5.1 RestAPI for executing Conductor operations.

It is possible to control Conductor in RestAPI.
The functions possible to control are the ones found in the (Conductor Menu) group-> (Conductor Execution) menu -> (Conductor Confirmation) menu.

Table 6-1. Target menu list

Menu group	Menu name	Menu ID
Conductor	Conductor Execution	2100180004
	Conductor Execution check	2100180005

5.1.1 Request format.

Execute HTTP Request with the information below

・Path
https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no=MenuID
Please refer to “Table 6-3. Parameters that can be specified to X-command list” for the Menu ID

・HTTP Header

Table 5-1 HTTP Header parameter list.

HTTP ヘッダ	説明
Method	POST only
Content-Type	Specifies “application/json”
Authorization	In order to access ITA Menus that requires authentication, concatenate Login ID and Password with half-width colon (:) and Base64econdoded value
X-Command	Choose between EXECUTE CANCEL SCRAM RELEASE

Table 5-2 Parameters that can be specified to X-command list

X-Command	Description	Screen	Menu ID
EXECUTE	Executes Conductor operation	Conductor Execution	2100180004
CANCEL	Deletes Conductor reservation.	Conductor confirmation	2100180005
SCRAM	Initiates emergency stop for Conductor.	Conductor confirmation	2100180005
RELEASE	Resumes paused Conductor.	Conductor confirmation	2100180005

In the following section, each X-command parameter will be explained.

5.1.2 **Response Items**

In the following section, the response items for executing X-commands will be explained.

Table 5-3 Response Item list

Item	Remarks
CONDUCTOR_INSTANCE_ID	Used to operate on Conductor instances..
NODE_INSTANCE_NO	Only used when RELEASE
RESULTCODE	Result codes for Command execution 000：Normal end 001: Cannot be executed 002：Cannot delete reservation 003：Cannot initiate Emergency stop 004：Cannot resume from pause
RESULTINFO	Detailed information

5.1.3 **EXECUTE**

Select desired Conductor class and Operation and execute. It is possible to specify the reservation time/date, Skip and Operation ID for each movement registered in Conductor class.

・Parameter

Specify the following as” content” in JSON format

Table 5-4 Individually specified Operation ID parameter list.

Parameter	Values
CONDUCTOR_CLASS_NO	Conductor class ID
OPERATION_ID	Operation ID
PRESERVE_DATETIME	Reservation Data/Time（YYYY/MM/DD tt:mm）

1）EXECUTE　Json Description example

When the Conductor class ID is 1, Operation ID is 1001 and the reservation date/time is 2016/01/01 00:00

▽Written in JSON Format

```
{
    "CONDUCTOR_CLASS_NO": 1,
    "OPERATION_ID": 1001,
    "PRESERVE_DATETIME": "2016/01/01 00:00",
}
```

Figure 6.1-1 EXECUTE　Json Description example

・Response

The return response is stored in JSON format.

```
{
    "status": "Execution success/failure",
    "resultdata": {
        "CONDUCTOR_INSTANCE_ID": "Execution No ",※
        "RESULTCODE": "Result Code",
        "RESULTINFO": "Detailed Information"
    }
}
```

※Use when controlling these instances after execution (INFO, CANCEL, SCRAM, RELEASE)

5.1.4 **CANCEL**

Cancels the specified Conductor ID that has a registered reservation date/time.

・Parameter

Specify the following as “content” in JSON format.

Table 5-5　Conductor Execution Cancel Parameter table.

Parameter	Value
CONDUCTOR_INSTANCE_ID	Conductor Instance ID※

※Obtained by EXECUTE return value.

・Response

Return responses are stored in JSON format. See below for information about items

```
{
    "status": "SUCCEED",
    "resultdata": {
        " CONDUCTOR_INSTANCE_ID": " Conductor ID when executed ",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed Information"
    }
}
```

5.1.5 **SCRAM**

Select Conductor and Node instance IDs and un-pause any points that are set to pause.

・Parameter

Specify the following as “content” in JSON format.

Table 5-6 Conductor Process Pause release Parameter table

Parameter	Value
CONDUCTOR_INSTANCE_ID	Conductor instance ID※

※Obtained by EXECUTE return value.

・Response

Return responses are stored in JSON format. See below for information about items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        " CONDUCTOR_INSTANCE_ID": " Conductor ID when executed ",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
    }
}
```

5.1.6 **RELEASE**

Select Conductor and Node instance ID and un-pause any points that are set to pause.

・Parameter

Specify the following as “content” in JSON format.

Table 5-7 Conductor process pause release parameter table

Parameter	Value
CONDUCTOR_INSTANCE_ID	Conductor instance ID※1
NODE_INSTANCE_ID	Node instance ID※2

※1 Obtained by EXECUTE Return value

※2 Refer to “6.2 RestAPI for Conductor confirmation” for obtaining Node Instance ID.

・Response

The returned response is stored in JSON format. See below for information regarding the items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "CONDUCTOR_INSTANCE_ID": "Conductor ID when executed",
        " NODE_INSTANCE_NO": "Node instance ID of (Conductor pause)",
        "RESULTCODE": "Result Code",
        "RESULTINFO": "Detailed information"
    }
}
```

5.2 RestAPI for Conductor confirmation

Conductor can be controlled from RestAPI。
The functions possible to control are the functions in (Conductor) Menu group-> (Conductor Execution) menu-> (Conductor Confirmation) menu.

Table 5-8 Target menu list

Menu group	Menu name	Menu ID
Conductor	Conductor confirmation	2100180005

5.2.7 **Request format**

Execute HTTP Request with the following information.

・Path

https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no=MenuID
For menu ID, please refer to “Table 5-9 List of parameters that can be specified for X-Command

・HTTP header

Table 5-10 HTTP header parameter list

HTTP header	Description
Method	POST only
Content-Type	Specify “application/json”.
Authorization	In order to access ITA menus that requires authentication, concatenate Login ID and Password with half-width colon (:) and Base64encoded value.
X-Command	Can choose the following: INFO

Parameters that can be specified to X-command.

Table 5-11 Parameters that can be specified for X-command list

X-Command	Description	Menu	Menu ID
INFO	Checks the status of the Conductor and returns it.	Conductor Confirmation	2100180005

In the following section, the response items for each running X-command will be explained.

5.2.8 **Response items**

In the following section, the response items for each running X-command will be explained.

Table 5-12 Response item list

Item	Remarks
CONDUCTOR_INSTANCE_ID	Use to operate on CONDUCTOR instances.
RESULTCODE	Command execution success/failure

	000：Normal end
RESULTINFO	Detailed information

5.2.9 **INFO**

Specify the instance ID when executing Conductor to obtain runtime information.

・Parameter

Specify the following as “content” in JSON format.

Table 5-13 Conductor Execution information acquisition parameter table

Parameter	Value
CONDUCTOR_INSTANCE_ID	Conductor Instance ID※

※Obtained by EXECUTE return value.

・Response

The return response is stored in JSON format.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "CONDUCTOR_INSTANCE_INFO": {
            "CONDUCTOR_INSTANCE_ID": 1,
                ・・・（abbr.）See (1) below for information about items・・・
        },
        "NODE_INFO": [
            "node-1": {
                "NODE_NAME": "11",
                    ・・・（abbr.）See (2) below for information about items・・・
            }
        }
        ・・・・
        ],
        "RESULTCODE": "000",
        "RESULTINFO": ""
    }
}
```

(1) Conductor instance information array stored in CONDUCTOR_INSTANCE_INFO

Table 5-14 Instance array table

Key	Content
CONDUCTOR_INSTANCE_ID	Conductor instance ID
CONDUCTOR_CLASS_NO	ID of the original class of this instance
STATUS_ID	Detailed execution status. More in the table below “Table 5-16”
EXECUTION_USER	Execution user
ABORT_EXECUTE_FLAG	Emergency stop flag. Not issued：1 Issued：2

OPERATION_NO_IDBH	Registered operation ID
OPERATION_NAME	Registered operation name
TIME_BOOK	Reserved time/date
TIME_START	Start time/date
TIME_END	End time/date

(1) Node instance information that are going to be stored in NODE_INFO

Table 5-15 Node instance information table

Key	Content
NODE_NAME	Node name
NODE_INSTANCE_NO	Node instance No
NODE_TYPE_ID	Node type ID. More in the table below Table 5-18”
STATUS	Node status. More in the table below “Table 5-19”
SKIP	Skip is set ：2
TIME_START	Start time/date
TIME_END	End time/date
OPERATION_ID	Individually specified Operation ID
OPERATION_NAME	Individually specified Operation Name

① Target ID Table

Table 5-16 Status ID of Conductor instance when running.

ID	Status
1	Not executed
2	Not executed (reserved)
3	Running
4	Running (Extended)
5	Normal end
6	Emergency stop
7	Abnormal end
8	Unexpected error
9	Delete reservation

Table 5-17 Orchestra ID table

ID	Status
3	Ansible Legacy
4	Ansible Pioneer
5	Ansible Legacy Role
10	Terraform

Table 5-18 Node type ID table

ID	Status
1	Conductor start
2	Conductor end
3	Movement
4	Conductor call
5	Parallel branch
6	Conditional branch
7	Parallel merge
8	Conductor pause
10	Symphony call

Table 5-19 Node instance runtime status.

ID	Status
1	Not executed
2	Preparing
3	Running
4	Running (extended)
5	Finished
6	Abnormal end
7	Emergency stop
8	On hold
9	Normal end
10	Preparation error
11	Unexpected error
12	Skip complete
13	On hold after skip
14	Skip finished

6 Movement

6.1 RestAPI for Movement execution

Operating Movement from RestAPI is possible.
The available function is same as the operation in “Execution” and “Check execution status” menu in the following menu group.

Table 6-1 Execution, Check execution status menu list

Menu group	Menu name	Menu ID
Ansible-Legacy	Execution	2100020111
	Check execution status	2100020112
Ansible-Pioneer	Execution	2100020211
	Check execution status	2100020212
Ansible-LegacyRole	Execution	2100020312
	Check execution status	2100020313
Terraform	Execution	2100080009
	Check execution status	2100080010

6.1.1 **Request type**

Send HTTP request with the following information.

・Path
https://<HostName>:<Port>/default/menu/07_rest_api_ver1.php?no=MenuID
Please refer to “Table 6-3 List of parameters that can be specified for X-Command” for Menu ID.

・HTTP Header

Table 6-2 HTTP header parameter list

HTTP Header	Description
Method	POST only
Content-Type	Specify “application/json”
Authorization	When accessing the menu that needs ITA authentication, specify the value of “Login ID” and “Password”* connected with half-width colon (:) then encoded with base64.
X-Command	EXECUTE CANCEL SCRAM These three can be chosen.

Parameters that can be specified for X-Command

Table 6-3 List of parameters that can be specified for X-Command

X-Command	Description	Target menu	Menu ID
EXECUTE	Schedule/Execute Movement	Execution	2100020111 2100020211 2100020312 2100080009
CANCEL	Cancel execution schedule	Check operation status	2100020112 2100020212 2100020313 2100080010
SCRAM	Perform emergency stop	Check operation status	

The following is the explanation of each X-command parameter.

6.1.2 **Response item**

The following is the explanation of the response items in each X-command parameter.

Table 6-4 Response item list

Item name	Remarks
EXECUTION_NO	Used to operate with Operation No.
RESULTCODE	The code of execution status 000: Normal end 001: Not able to perform execution 002: Not able to cancel schedule 003: Not able to perform emergency stop
RESULTINFO	Detailed information

6.1.3 **EXECUTE**

Specify the Movement class and Operation then perform execution. Specifying scheduled execution date/time and execution mode (Dry run/Execute) is possible.

・Parameter

Please specify the following items in JSON format to “content”.

Table 6-5 Movement execution parameter list

Parameter name	Value
MOVEMENT_CLASS_ID	Movement class ID
OPERATION_ID	Operation ID
PRESERVE_DATETIME	Scheduled execution date/time (YYYY/MM/DD tt:mm)
RUN_MODE	1:Execute 2:Dry run

Example) JSON description example

```
{
    "MOVEMENT_CLASS_ID": 1,
    "OPERATION_ID": 1,
    "PRESERVE_DATETIME": "2019/12/24 15:44",
    "RUN_MODE": 1
}
```

・Response

The returned response is stored in JSON format. Please refer to the following for the response items.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "EXECUTION_NO": "Operation No",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
    }
}
```

6.1.4 **CANCEL**

Specify the Operation No. of the registered Operation whose execution date is scheduled and cancel the schedule.

・Parameter

Please specify the following items in JSON format to “content”.

Table 6-6 Movement execution parameter list

Parameter name	Value
EXECUTION_NO	Execution No※

※The value obtained from the return value of EXECUTE.

・Response

The returned response is stored in JSON format.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "EXECUTION_NO": "Operation No",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
    }
}
```

6.1.5 **SCRAM**

Specify the Operation No. of the executing Operation and perform emergency stop.

・Parameter

Please specify the following items in JSON format to “content”.

Table 6-7 Movement execution parameter list

Parameter name	Value
EXECUTION_NO	Operation No※

※The value obtained from the return value of EXECUTE.

・Response

The returned response is stored in JSON format.

```
{
    "status": "SUCCEED",
    "resultdata": {
        "EXECUTION_NO": "Operation No",
        "RESULTCODE": "Result code",
        "RESULTINFO": "Detailed information"
    }
}
```

7 Appendix

7.1 Troubleshooting

No	Content
Q-1	Uploading files using RestAPI takes too long. An error occurs during the registration process when using RestAPI. The display and operation web screen becomes slow when uploading files with RestAPI.
A-1	It is possible that your PHP memory settings are set too low. Please check the values of the following parameters in the PHP configuration file (php.ini) and increase the set maximum value. ・memory_limit　Memory available for PHP allocation ・post_max_size　Maximum size allowed for post data.