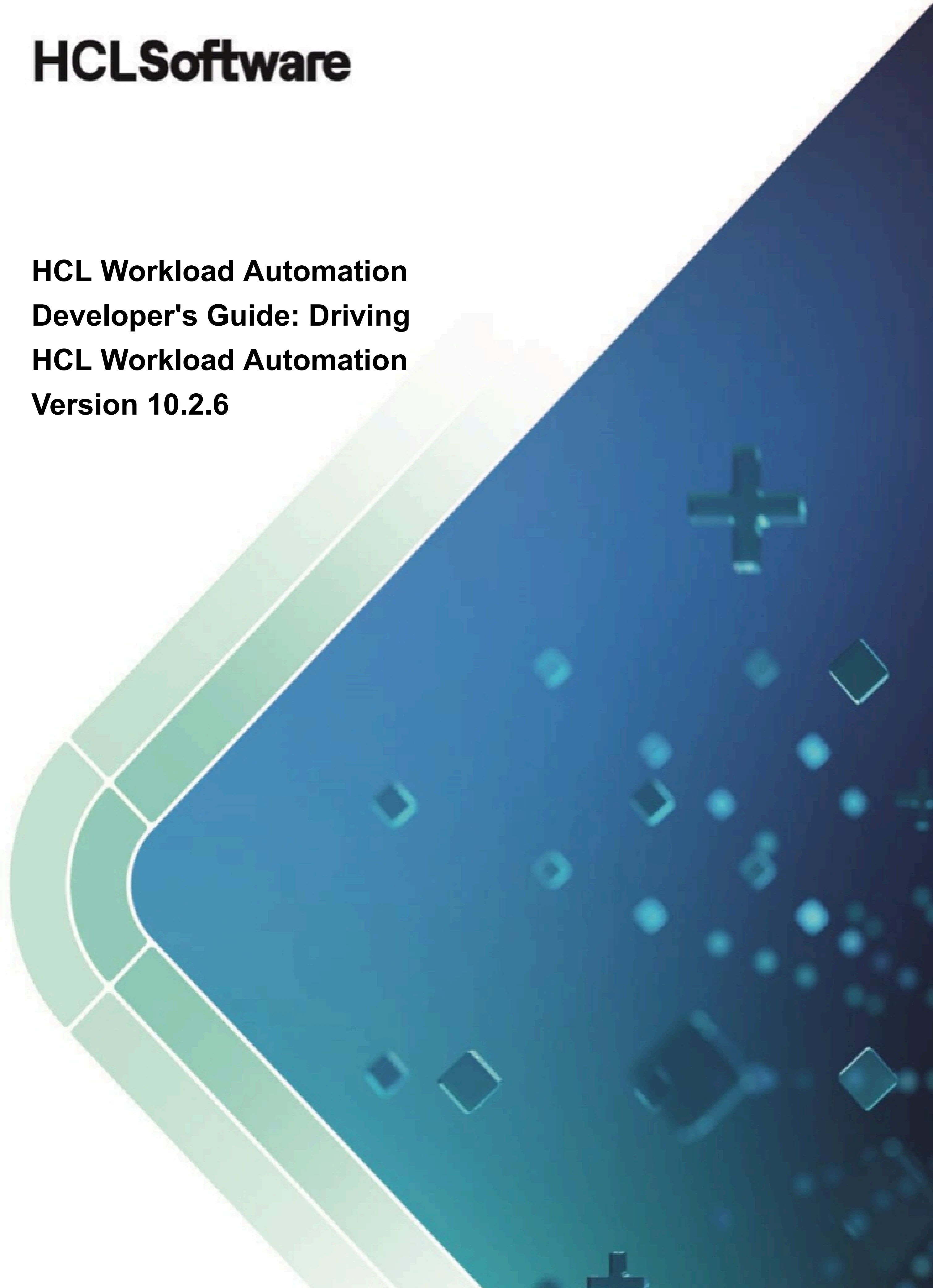
HCLSoftware

HCL Workload Automation Developer's Guide: Driving HCL Workload Automation

Version 10.2.6



Note

Before using this information and the product it supports, read the information in Notices on page xxv.

This edition applies to version 10, release 2, modification level 6 of HCL Workload Automation (program number 5698-T09) and to all subsequent releases and modifications until otherwise indicated in new editions.

Contents

About this guide

Provides an overview of the guide, with information about changes made to it since the last release, and who should read it. It also supplies information about obtaining resources and support from HCL.

Developer's Guide: Driving HCL Workload Automation introduces you to the application programming interfaces available to drive HCL Workload Automation products from your own applications.

What is new in this release

Learn what is new in this release.

For information about the new or changed functions in this release, see *Overview*, section *Summary of enhancements*.

New or changed content is marked with revision bars.

Chapter 1. Introduction to driving HCL Workload Automation

Provides an overview of the entire publication.

Developer's Guide: Driving HCL Workload Automation describes the application programming interfaces which you can use to drive HCL Workload Automation products from your own applications.

Use the REST API application programming interface to create your own GUI or command-line interface to perform all the functions of the command-line programs composer, conman, and planman and the Dynamic Workload Console. This includes performing the following tasks in HCL Workload Automation and HCL Workload Automation for Z:

- Modifying objects in the database
- Submitting workload
- Monitoring the plan
- Performing actions on the plan, such as remedial actions in the event that a job fails

The information about the application programming interfaces is organized as follows:

- Driving HCL Workload Automation with REST API on page 7

Chapter 2. Driving HCL Workload Automation with REST API

HCL Workload Automation provides a set of fully functional APIs that are implemented based on Representational State Transfer (REST) services. REST APIs let you easily integrate our workload scheduling capabilities with heterogeneous third party environments into complex automated systems. The same product functionalities covered by J2EE APIs are available with REST APIs. REST APIs use an independent programming language and enable easier network configuration and firewall traversal. The following are some examples or scenarios where APIs can be implemented:

- You can develop your own graphical interface to create and modify scheduling definitions and update items in the plan.
- You can update definitions in the database or items in the plan within a script for integration or automation.
- When a specific event occurs within an external product, you can automatically submit a batch workload through HCL Workload Automation.
- In a managed file transfer solution, when a specific file arrives, you can submit one or more jobs to elaborate the file, closing the loop on your business process, whether it be bank transactions, a payroll process, or report generation. Your external managed file transfer product starts the business process and HCL Workload Automation takes care of the processing, assuring that it is monitored together with the rest of the processes from a single point of control.

The HCL Workload Automation REST API provides several services to administer engines, event rules, workload modelling, plans, and security.

REST API V2

REST API V2 have been implemented and are easier to configure, more powerful and flexible. It is highly recommended to use them for any future integration.

Accessing REST APIs

After installing your master domain manager or backup master domain manager, you can access the available REST API services by connecting to the following URL:

```
https://hostname:port_number/twsd
```

where,

hostname

The hostname of the master domain manager or the backup master domain manager.

port_number

The HTTPS port number of the master domain manager or backup domain manager. The default is 31116.

You can try out the REST API services and the operations available for each API on Swagger Docs by connecting to `https://MDM_IP_address:tdwbport/twsd/`. There, click **List Operations** to view the operations available with the service, and then click **Expand Operations** to view details such as, the implementation notes, parameters, and response messages for each method. At the end of the details you can find a **Try it out!** button to see the operation in action.

You can also access some HCL Workload Automation REST API samples here: REST API samples.

REST API - creating a new job definition in the database

Use REST APIs to add a new job definition to the database.

About this task

You can create a job definition by submitting the definition payload through a POST REST API.

1. Prepare the payload template for the new job definition. Modify the fields within the **def** array with the required values.

   ```
   {
       "kind": "JobDefinition",
       "def": {
           "folder": "/",
           "name": "JOB_TEST",
           "workstation": "/RMMYCLDTL16961",
           "type": "UNIX",
           "task": {
               "unix": {
                   "taskString": "ls",
                   "userName": "twsuser",
                   "isCommand": true
               }
           }
       }
   }
   ```

   To retrieve supporting information, such as the name of the workstation you want to use, utilize the respective GET endpoint.
2. Use the POST/twsd/api/v2/model/jobdefinition endpoint to make a POST request.
3. In the **Swagger UI**, expand the endpoint, paste the edited payload, and click **Execute**.

   Result

   The API response returns the ID for the new job definition.
4. Verify that the job definition was created.
   a. Run the GET/twsd/api/v2/model/jobdefinition API call again.
   b. Check that the job definition is present in the results list.

REST API - setting a job stream in draft

Use REST APIs to change the status of a job stream to *draft*.

About this task

This process involves retrieving the job stream current data through a GET REST API and then submitting an updated version through a POST REST API.

1. First, you need to get the unique ID and the current configuration (payload) of the job stream you want to modify. Use the GET/twsd//api/v2/model/jobstream endpoint to find the job stream.

    a. In the **Swagger UI**, expand the endpoint, select **Filter by exact name and matching folder and sort results in name descending order** in the **oql string** parameter field, enter the name of the job stream in the name field, and click **Execute**.

    b. From the API response, copy and save the following two items: the job stream ID in the response header and the entire JSON payload from the response body. Example of a response payload:

```
{
 "count": 1,
 "results": [
   {
     "kind": "JobStream",
     "key": "/WA_FTA_XA#/FINAL",
     "def": {
       "id": "9254f5da-181a-3be8-a1f1-8c978848c261",
       "abstractId": "61918471-ce1b-3d21-bb66-6dd4d9e3a5ba",
       "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
       "folder": "/",
       "name": "FINAL",
       "workstationId": "9819daa5-1a2a-385b-b238-cfa7309c65d2",
       "workstation": "/WA_FTA_XA",

...
}
```

2. Now, use the information you just retrieved to send an update request. Use the PUT/twsd/api/v2/model/jobstream/{jobStream_id} endpoint to update the job stream.

    a. In the Swagger UI, expand the endpoint and paste the Job Stream ID you copied in the previous step in the **ID** parameter field.

    b. In the **Request body** field, paste the entire JSON payload you copied.

    c. In the JSON you just pasted, find the **options** parameter and include the **draft** parameter in the options list:

```
...

"validTo": "2025-09-23T09:44:10.278Z",
    "timeZone": "string",
    "description": "This is a test job stream.",
    "documentation": "string",
    "options": [
      "draft"
    ],
...
```

    d. Click **Execute**. The API returns a success message, and your job stream is now in draft mode.

3. To confirm the job stream is now in draft mode, run the GET/twsd//api/v2/model/jobstream API call again and check that the **draft** value is present in the **option** list.

REST API - submitting a job stream in the current plan

Use REST APIs to submit a job stream in the plan.

About this task

This process involves retrieving a job stream's ID through a GET REST API and then submitting it into the current plan through a POST REST API.

1. First, get the unique ID of the job stream you want to submit. Use the GET/twsd/api/v2/model/jobstream endpoint to find your job stream.

   You can query for the job stream using either model filters or OQL. The following substeps show how to retrieve a job stream named *JS-API*.

   a. **To query with model filters:** use a syntax analogous to the composer syntax. For the example job stream, use the following syntax: `/@/@#/@/JS-API`.

   b. **To query with OQL:** from the dropdown menu, select **Filter by exact name and matching folder and sort results in name descending order** in the **OQL string** parameter field. Enter the name of the job stream in the name field to obtain the following query: `name = 'JS-API' AND folder LIKE '/' ORDER BY name DESC`.

   c. After selecting a query method, click **Execute**.

   d. From the API response, copy and save the job stream **id**.

   Example of a response payload:

```
{
  "count": 1,
  "results": [
    {
      "kind": "JobStream",
      "key": "/WA_DA#/JS-API",
      "def": {
        "id": "26dc6d8d-5cff-4002-8346-8027ea50f81c",
        "abstractId": "26300eb5-547c-46a1-b450-fab39fe0d4a9",
        "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
        "folder": "/",
        "name": "JS-API",
        "workstationId": "f2e69991-a64a-31d9-a006-dccc76ff95c0",
        "workstation": "/WA_DA",
        "options": [],
        "saturdayIsFree": true,
        "sundayIsFree": true,
        "runCycles": [],
        "exclusiveRunCycles": [],
        "asap": true,
        "perJobLatestStart": false,
        "matchingCriteria": {
          "type": "sameDay"
          }
        }
      }
    ]
  }
```

2. Now, use the ID you retrieved to submit the job stream with the POST/twsd/api/v2/plan/jobstream/{model_jobstream_id}/submit endpoint.
   a. In the **Swagger UI**, expand the endpoint and paste the job stream ID into the **ID** parameter field.
   b. In the **Request body** field, delete the entire content. No request body is required for this action.
   c. Click **Execute**.

   Result

   The API returns a success message with the ID of the job stream in the plan. Your job stream is now scheduled to run in the current plan.
3. To verify the submission, run the GET/twsd/api/v2/plan/jobstream API endpoint. Confirm that the response body contains the JSON object for the job stream you submitted.

REST API - setting a workstation fence

Use REST APIs to change the fence value of a workstation in the plan.

About this task

This process outlines the steps required to modify a workstation fence value within the current plan. The fence value plays a crucial role in job scheduling management, as jobs are not initiated on a workstation if their priorities are at or below the job fence value.

1. To change the fence value of a workstation in the plan, use the POST/twsd/api/v2/plan/workstation/action/update-fence endpoint. In the **Swagger UI**, expand the endpoint, identify the workstation you want to modify by querying with plan filters or OQL, modify the **Fence** parameter with the desired fence value, and then click **Execute**.

   Result

   If the command is successful, a response with the ID of the modified workstation in the plan is displayed, as shown in the following example:

   ```
   {
     "successfulResourceResponses": [
       {
         "id": "75196d79-50b0-345c-9308-ee4d69efe147",
         "error": false
       }
     ],
     "failedResourceResponses": []
   }
   ```
2. To verify that the fence value was updated, run the GET/twsd/api/v2/plan/workstation API call and check that the **Fence** value in the response is equal to the value you previously set.

REST API - submitting a job with dependencies in the current plan

Use REST APIs to submit a job with a single dependency in the plan.

About this task

This process involves retrieving the unique IDs of two jobs (a successor and a predecessor) and then creating a dependency between them using a POST REST API.

1. First, get the unique ID of the successor job. Use the GET/twsd/api/v2/model/jobdefinition endpoint to find its definition in the database.

   You can query for the job definition using either model filters or OQL. The following substeps show how to retrieve a job definition named *TESTJOB*.

   a. **To query with model filters:** use a syntax analogous to the composer syntax, for example: `/@/@#/@/TESTJOB`.

   b. **To query with OQL:** from the dropdown menu, select **Filter by exact name and matching folder and sort results in name descending order** in the **OQL string** parameter field. Enter the name of the job in the name field to obtain the following query: `name = 'TESTJOB' AND folder LIKE '/TEST/' ORDER BY name DESC`.

   c. After selecting a query method, click **Execute**.

   d. From the API response, copy and save the job definition **id**.

      Example of a response payload:

```
{
  "count": 1,
  "results": [
    {
      "kind": "JobDefinition",
      "key": "/RMMYCLDTL15094_1#/TEST/TESTJOB",
      "def": {
        "id": "e7ca3702-2411-32c0-bef7-e425f0a2f815",
        "folderId": "1f3fbed7-7c1f-3eb0-95c1-80f1ff2815b0",
        "folder": "/TEST/",
        "name": "TESTJOB",
        ...
      }
    }
  ]
}
```

2. Next, get the data for the predecessor job. Use the GET/twsd/api/v2/plan/job endpoint to find the job currently in the plan.

   a. Use the query filters or OQL to locate the job you want to use as a predecessor.

   b. Click **Execute** and, from the response, copy and save the predecessor job data, including its **id**.

Example of a response payload for a predecessor job:

```
{
  "count": 1,
  "results": [
    {
      "flowNodeType": "JOB",
      "id": "a8ca4deb-b23d-33f3-af22-bf9b53802da4",
      "planId": "718ccf91-7ee5-3a8a-8431-e4e3252cac93",
      "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
      "folder": "/",
      "name": "LS1548259130",
      "workstationId": "4ffdee00-5e6d-3463-9bad-1ab7edac6d90",
      "workstation": "/RMMYCLDTL15094_1",
      "cpuFolderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
      "jobStreamId": "f2616f78-924a-36d8-aa2f-f65299a0ed54",
      "jobStreamName": "JOBS",
      "jobStreamWorkstationId": "4ffdee00-5e6d-3463-9bad-1ab7edac6d90",
      "jobStreamWorkstation": "/RMMYCLDTL15094_1",
      "position": 2,
      "key": "RMMYCLDTL15094_1;JOBS;LS1548259130",
      "jobDefinitionFlags": {},
      "jobStatusFlags": {},
      "jobOptions": {
        "every": false
      },
      "jobDefinition": {
        "name": "LS1548259130",
        "workstationId": "4ffdee00-5e6d-3463-9bad-1ab7edac6d90",
        "workstation": "/RMMYCLDTL15094_1",
        "cpuFolderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
        "taskType": "UNIX",
        "task": {
          "UNIX": {
            "taskString": "ls",
            "userName": "twsuser",
            "isCommand": true
          }
             ...
            }
            ]
            }
```

3. Now, use the data you retrieved to add the dependency with the POST/twsd/api/v2/plan/job/{job_id}/action/add-dependencies endpoint.

    a. In the **Swagger UI**, expand the endpoint and paste the ID of the successor job (from the first step) into the **ID** parameter field in the path.

    b. In the **Request body** field, paste the following content, using the IDs you copied from the previous steps. The first **id** belongs to the successor job, and the **jobId** belongs to the predecessor job.

```
{
                                "dependencies": [
                                {
                                "dependencyType": "INTERNAL",
                                "id": "e7ca3702-2411-32c0-bef7-e425f0a2f815",
                                "dependencyStatus": "SATISFIED",
                                "mandatory": true,
                                "conditional": true,
                                "critical": true,
                                "statusConditions": [
                                "STARTED"
                                ],
                                "outputConditions": [
                                "string"
                                ],
                                "jobStreamName": "JOBS",
                                "jobStreamWorkstation": "RMMYCLDTL15094_1",
                                "jobStreamSchedTime": "2025-09-30T10:20:43.212Z",
                                "jobStreamFolder": "/",
                                "jobId": "a8ca4deb-b23d-33f3-af22-bf9b53802da4",
                                "jobName": "LS1548259130",
                                ...
                                }
                                ]
                                }
```

c. Click **Execute**.

Result

The API returns a success message, and the dependency is now active in the current plan.

4. To verify the dependency was added, run the GET/twsd/api/v2/plan/job/{job_id} API endpoint, using the ID of the successor job. Examine the **dependencies** object in the response body to confirm it is correct.

REST API - submitting an ad-hoc job

Use REST APIs to submit an ad-hoc job.

About this task

This process involves submitting an ad-hoc job through a POST REST API.

1. Use the POST /twsd/api/v2/plan/job/submit-ad-hoc-job API endpoint.
2. In the request body, define the **task** and the **workstationKey**.

   The following examples show different task definitions:

   Simple Task

```
{
  "task": {
    "UNIX": {
      "taskString": "ls",
      "isCommand": "true",
      "userName": "wauser"
    }
  }
}
```

Executable Task (Corrected)

```
{
  "task": {
    "OTHER": {
      "taskString": "&lt;?xml version=\"1.0\" encoding=\"UTF-8\"?>\n&lt;jsdl:jobDefinition
 xmlns:jsdl=\"http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdl\"
 xmlns:jsdle=\"http://www.ibm.com/xmlns/prod/scheduling/1.0/jsdle\" name=\"executable\">\n
    &lt;jsdl:application name=\"executable\">\n        &lt;jsdle:executable
 interactive=\"false\">\n            &lt;jsdle:script>ls&lt;/jsdle:script>\n
 &lt;/jsdle:executable>\n    &lt;/jsdl:application>\n&lt;jsdl:jobDefinition>",
      "userName": "wauser"
    }
  }
}
```

The following example shows a complete JSON request body:

```
{
  "workstationKey": "WA_AGT",
  "task": {
    "UNIX": {
      "taskString": "ls",
      "isCommand": "true",
      "userName": "wauser"
    }
  }
}
```

3. Click **Execute**.

   Result

   The API returns a success message, and the ad-hoc job is submitted.
4. To verify the submission, run the GET /twsd/api/v2/plan/job API call. Check that the ad-hoc job is successfully added to the plan.

REST API - retrieving the predecessors of a job stream in the plan

Use REST APIs to retrieve the list of a job stream predecessors.

About this task

This process involves retrieving a job stream data from the plan and then inspecting its **dependencies** field to identify all its predecessors.

1. First, get the data for the job stream for which you want to find predecessors. Use the GET/twsd/api/v2/plan/jobstream endpoint.

    a. In the **Swagger UI**, expand the endpoint. Use the filters or the **oql string** parameter to locate your target job stream.

    b. Click **Execute**.

    c. From the API response, locate the **dependencies** array in the response body. This array contains the IDs of all predecessor jobs and job streams.

    Example of a response payload showing the dependencies array:

```
JSON

{
  "flowNodeType": "JOBSTREAM",
  "id": "9daec769-39e0-3283-9006-2933c84d3c78",
  "planId": "57c9a3d5-e6a0-3f0d-96c4-c65602a069b2",
  ...
  "dependencies": [
    {
      "dependencyType": "EXTERNAL_JOB",
      "id": "612f1cd8-89a8-30c0-ab27-da8f5c89b0d3",
      "dependencyStatus": "UNDECIDED",
      "jobId": "612f1cd8-89a8-30c0-ab27-da8f5c89b0d3",
      "jobName": "SWITCHPLAN",
      "jobStreamId": "659aa50d-8128-3d11-9977-8390004c2a06",
      "jobStreamName": "FINAL",
      "jobStreamWorkstation": "/RMMYCLDTL4267_XA",
      "jobStreamSchedTime": "2025-10-13T21:59:00Z",
      "jobStreamFolder": "/",
      "folderOriginalName": "RMMYCLDTL4267_XA",
      "jobStreamSuccId": "9daec769-39e0-3283-9006-2933c84d3c78",
      "pending": false,
      "zombie": false
    }
  ],
  "dependenciesStats": {
    "numberOfDependencies": 1,
    "numberOfJobDependencies": 1,
    "numberOfJobStreamDependencies": 0,
    "numberOfInternetworkDependencies": 0,
    "numberOfPromptDependencies": 0,
    "numberOfResourceDependencies": 0,
    "numberOfFileDependencies": 0,
    "numberOfUnresolvedDependencies": 1,
    "numberOfNonResourceUnresolvedDependencies": 1,
    "numberOfCompletedDependencies": 0,
    "numberOfSuccessors": 0
  },
  ...
}
```

2. For each entry in the **dependencies** array, use its **jobId** or **jobStreamId** to retrieve the full details of that predecessor.
   a. To get a predecessor job details, use the GET/twsd/api/v2/plan/job/{job_id} endpoint, pasting the **jobId** from the array into the ID parameter.
   b. To get a predecessor job stream details, use the GET/twsd/api/v2/plan/jobstream/{jobstream_id} endpoint, pasting the **jobStreamId** from the array into the ID parameter.
   c. Click **Execute**.

   Result

   The response for each call contains the detailed information for that specific predecessor. Repeat this process for every dependency listed in the array to get the complete list of predecessors.

REST API - retrieving the successors of a job stream in the plan

Use REST APIs to retrieve the list of a job stream successors.

About this task

This process involves finding the unique ID of a job stream in the plan and then using that ID to query for other jobs and job streams that depend on it.

1. First, get the unique ID of the job stream for which you want to find successors. Use the GET/twsd/api/v2/plan/jobstream endpoint to find your job stream.

   a. In the **Swagger UI**, expand the endpoint. Use the filters or the **oql string** parameter to locate the job stream.

   b. Click **Execute**.

   c. From the API response, copy and save the job stream **id**.

   Example of a response payload:

```
{
  "flowNodeType": "JOBSTREAM",
  "id": "9daec769-39e0-3283-9006-2933c84d3c78",
  "planId": "57c9a3d5-e6a0-3f0d-96c4-c65602a069b2",
  "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
  "folder": "/",
  "name": "FINALPOSTREPORTS",
  "workstationId": "af930748-42c6-3440-be71-4d9ee1050130",
  "workstation": "/RMMYCLDTL4267_XA",
  ...
}
```

2. Now, use the ID you copied to find all successors. You must query two endpoints to get a complete list of both jobs and job streams that depend on your target job stream.

a. To find successor jobs, use the GET/twsd/api/v2/plan/job endpoint. In the **oql** parameter field, enter the following query, replacing the example ID with your job stream ID:

```
dependencies.jobStreamId = '9daec769-39e0-3283-9006-2933c84d3c78'
```

b. To find successor *job streams, use the GET/twsd/api/v2/plan/jobstream endpoint with the same **oql** filter.

c. Click **Execute** for each request.

Results

The API responses for each call contain the lists of jobs and job streams that are successors to your target job stream.

REST API - listing and releasing job stream dependencies in the plan

Use REST APIs to list the dependencies of a job stream and then release or remove them.

About this task

This process involves retrieving a job stream current data and list of dependencies through a GET REST API, and then managing those dependencies using a PUT REST API.

1. First, get the current data and dependencies for the job stream you want to modify. Use the GET/twsd/api/v2/plan/jobstream endpoint to find the job stream.

   a. In the **Swagger UI**, expand the endpoint. Use the query filters or the **oql string** parameter to locate your job stream.

   b. Click **Execute**.

   c. From the API response, copy the job stream **id** and examine the **dependencies** array in the response body to identify the dependencies you want to manage.

   Example of a response payload showing various dependency types:

```
{
  "flowNodeType": "JOBSTREAM",
  "id": "64ac512a-c2be-3537-9ca8-cfff3d1e9ded",
  "planId": "57c9a3d5-e6a0-3f0d-96c4-c65602a069b2",
  "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
  "folder": "/",
  "name": "TESTJS3",
  "workstationId": "da0d42a6-d0a4-32de-b106-aa35a87f1a8e",
  "workstation": "/RMMYCLDTL42671_1",
  "dependencies": [
    {
      "dependencyType": "JOIN",
      "satisfied": false,
      "joinName": "JOIN",
      "joinQuantity": 0,
      "members": [
        {
          "dependencyType": "EXTERNAL_JOBSTREAM",
          "id": "9daec769-39e0-3283-9006-2933c84d3c78",
          "dependencyStatus": "UNDECIDED",
          "satisfied": false,
          "depSequence": 1000,
          "conditional": true,
          "jobStreamId": "9daec769-39e0-3283-9006-2933c84d3c78",
          "jobStreamName": "FINALPOSTREPORTS",
          "jobStreamWorkstation": "/RMMYCLDTL4267_XA",
          "jobStreamSchedTime": "2025-10-13T21:59:00Z",
          "jobStreamFolder": "/",
          "predecessorStatus": {
            "cancelPending": false,
            "canceled": false,
            "started": false,
            "error": false,
            "dontRun": false,
            "status": "HOLD"
          },
          "folderOriginalName": "RMMYCLDTL4267_XA",
          "jobStreamSuccId": "64ac512a-c2be-3537-9ca8-cfff3d1e9ded",
          "statusConditions": [],
          "outputConditions": [],
          "pending": false,
          "zombie": false
        }
      ],
      "criteria": "ALL"
    },
    {
      "dependencyType": "RESOURCE",
      "id": "69b358f5-cc55-36cf-974e-51650820b292",
      "dependencyStatus": "UNDECIDED",
      "satisfied": false,
      "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
      "folder": "/",
      "name": "TESTR",
      "originalName": "TESTR",
      "workstation": "/RMMYCLDTL426711",
      "workstationId": "c598409a-a2de-3863-8e8f-8b36180c2dc3",
      "workstationOriginalName": "RMMYCLDTL426711",
      "quantity": 1,
      "available": 1,
      "actionOnComplete": "DEFAULT",
      "loadedFromDb": false
    },
    {
      "dependencyType": "PROMPT",
      "id": "3e5c3918-af8c-3d32-a733-b009492ee5a9",
      "dependencyStatus": "SATISFIED",
      "satisfied": true,
      "depSequence": 1,
      "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
      "folder": "/",
      "promptName": "TESTPRO",
      "promptOriginalName": "TESTPRO",
      "text": "test",
      "promptStatus": "ANSWERED_YES",
      "promptType": "Predefined Prompt",
      "number": 2,
      "loadedFromDb": false,
      "actions": [
        {
          "key": "REPLY_PROMPT",
          "id": "3e5c3918-af8c-3d32-a733-b009492ee5a9"
        }
      ]
    }
  ],
  "dependenciesStats": {
    "numberOfDependencies": 2,
    "numberOfJobDependencies": 0,
    "numberOfJobStreamDependencies": 1,
    "numberOfInternetworkDependencies": 0,
    "numberOfPromptDependencies": 0,
    "numberOfResourceDependencies": 1,
    "numberOfFileDependencies": 0,
    "numberOfUnresolvedDependencies": 2,
    "numberOfNonResourceUnresolvedDependencies": 1,
    "numberOfCompletedDependencies": 0,
    "numberOfSuccessors": 0
  },
  "timeRestrictions": {
    "startTime": "2025-10-14T12:30:00Z",
    "lateStartAction": "doNotRun",
    "timeDependent": true
  }
}
```

2. Now, use the job stream ID to update the dependencies by choosing one of the following actions.
    a. To **release all** dependencies, use the PUT/twsd/api/v2/plan/jobstream/{jobStream_id}/action/release-all-dependencies endpoint. In the **Swagger UI**, paste the job stream ID into the **ID** parameter. No request body is required.
    b. To release specific dependencies, including time dependencies, use the PUT/twsd/api/v2/plan/jobstream/{jobStream_id}/action/release-dependencies endpoint. Provide a request body containing a **dependencies** array with only the dependencies you want to release.
    c. To remove specific dependencies, use the PUT/twsd/api/v2/plan/jobstream/{jobStream_id}/action/remove-dependencies endpoint. Provide a request body containing a **dependencies** array with only the dependencies you want to remove.

Example of a request body for releasing or removing specific dependencies:

```
{
  "dependencies": [
    {
      "dependencyType": "JOIN",
      "satisfied": false,
      "joinName": "JOIN",
      "joinQuantity": 0,
      "members": [
        {
          "dependencyType": "EXTERNAL_JOBSTREAM",
          "id": "9daec769-39e0-3283-9006-2933c84d3c78",
          "dependencyStatus": "UNDECIDED",
          "satisfied": false,
          "depSequence": 1000,
          "conditional": true,
          "jobStreamId": "9daec769-39e0-3283-9006-2933c84d3c78",
          "jobStreamName": "FINALPOSTREPORTS",
          "jobStreamWorkstation": "/RMMYCLDTL4267_XA",
          "jobStreamSchedTime": "2025-10-13T21:59:00Z",
          "jobStreamFolder": "/",
          "predecessorStatus": {
            "cancelPending": false,
            "canceled": false,
            "started": false,
            "error": false,
            "dontRun": false,
            "status": "HOLD"
          },
          "folderOriginalName": "RMMYCLDTL4267_XA",
          "jobStreamSuccId": "64ac512a-c2be-3537-9ca8-cfff3d1e9ded",
          "statusConditions": [],
          "outputConditions": [],
          "pending": false,
          "zombie": false
        }
      ],
      "criteria": "ALL"
    },
    {
      "dependencyType": "RESOURCE",
      "id": "69b358f5-cc55-36cf-974e-51650820b292",
      "dependencyStatus": "UNDECIDED",
      "satisfied": false,
      "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
      "folder": "/",
      "name": "TESTR",
      "originalName": "TESTR",
      "workstation": "/RMMYCLDTL426711",
      "workstationId": "c598409a-a2de-3863-8e8f-8b36180c2dc3",
      "workstationOriginalName": "RMMYCLDTL426711",
      "quantity": 1,
      "available": 1,
      "actionOnComplete": "DEFAULT",
      "loadedFromDb": false
    },
    {
      "dependencyType": "PROMPT",
      "id": "3e5c3918-af8c-3d32-a733-b009492ee5a9",
      "dependencyStatus": "SATISFIED",
      "satisfied": true,
      "depSequence": 1,
      "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
      "folder": "/",
      "promptName": "TESTPRO",
      "promptOriginalName": "TESTPRO",
      "text": "test",
      "promptStatus": "ANSWERED_YES",
      "promptType": "Predefined Prompt",
      "number": 2,
      "loadedFromDb": false,
      "actions": [
        {
          "key": "REPLY_PROMPT",
          "id": "3e5c3918-af8c-3d32-a733-b009492ee5a9"
        }
      ]
    }
  ]
}
```

3. After choosing an endpoint and providing the required parameters, click **Execute**.

   Result

   The API returns a success message, and the job stream dependencies are updated.
4. To verify the update, run the GET/twsd/api/v2/plan/jobstream API call again for the same job stream. Check that the **dependencies** array in the response reflects the changes you made.

REST API - retrieving the predecessors of a job stream from the database

Use REST APIs to retrieve the list of a job stream predecessors from the database.

About this task

This process involves retrieving a job stream definition through a GET REST API and then inspecting its various dependency arrays, such as **externalPredecessors**, in the response body.

1. First, get the definition of the job stream for which you want to find predecessors. Use the GET/twsd/api/v2/model/jobstream endpoint.

    a. In the **Swagger UI**, expand the endpoint. Use the query filters or the **oql string** parameter to locate your target job stream.

    b. Click **Execute**.

    c. From the API response, locate the various dependency arrays within the definition, such as **externalPredecessors** and **promptDependencies**. These arrays contain the identifiers of the predecessor objects.

    Example of a response payload showing dependency arrays:

```
{
  "kind": "JobStream",
  "key": "/RMMYCLDTL42671_1#/TESTJS3",
  "def": {
    "id": "0168fa4e-3d7e-390c-a58d-49ca32fabc78",
    "abstractId": "e8cb402b-d1a6-3618-97d7-d0f3eb309796",
    "folderId": "742ba6a6-31aa-3051-b810-3ced4fc624b4",
    "folder": "/",
    "name": "TESTJS3",
    "workstationId": "62420b07-726c-3f5a-ae3a-df9a170c7626",
    "workstation": "/RMMYCLDTL42671_1",
    "options": [],
    "saturdayIsFree": true,
    "sundayIsFree": true,
    "runCycles": [],
    "exclusiveRunCycles": [],
    "asap": true,
    "perJobLatestStart": false,
    "matchingCriteria": {
      "type": "sameDay"
    },
    "externalPredecessors": [
      {
        "jobStreamAbstractId": "74795642-e468-3d11-90ed-c440478f8c56",
        "workstation": "/RMMYCLDTL42671_1",
        "jobStream": "/TESTJS2"
      },
      {
        "jobAbstractId": "c11b99fc-47d7-3f5c-927b-bafb8302c9fd",
        "workstation": "/RMMYCLDTL42671_1",
        "jobStream": "/TESTJS2",
        "jobName": "ALELS"
      }
    ],
    "joinConditions": [],
    "interNetworkDependencies": [],
    "promptDependencies": [],
    "fileDependencies": [],
    "resourceDependencies": [],
    "jobs": [
      {
        "id": "3317da2c-c5e8-3f89-b8a1-ec5082844894",
        "abstractId": "d67e427b-3820-3ec0-a4f4-abff2facf665",
        "name": "ALELS",
        "externalPredecessors": [
          {
            "jobAbstractId": "fe50e5c7-9084-32d2-a9e0-cf8e33c5ca47",
            "workstation": "/RMMYCLDTL42671_1",
            "jobStream": "/TESTJS",
            "jobName": "SLEEP"
          }
        ],
        "internalPredecessors": [],
        "joinConditions": [],
        "interNetworkDependencies": [],
        "promptDependencies": [
          {
            "globalPromptId": "5ab0a0cf-094a-3d4d-a94a-f059e0dae65d",
            "globalPrompt": "TESTP"
          }
        ],
        "fileDependencies": [],
        "resourceDependencies": []
      }
    ]
  }
}
```

2. For each predecessor identified in the dependency arrays, use its unique ID to retrieve its full definition.
    a. To get a predecessor job definition, use the GET/twsd/api/v2/model/jobdefinition/{job_abstract_id} endpoint, pasting the **jobAbstractId** into the ID parameter.
    b. To get a predecessor job stream definition, use the GET/twsd/api/v2/model/jobstream/{jobstream_abstract_id} endpoint, pasting the **jobStreamAbstractId** into the ID parameter.
    c. Click **Execute** for each request.

    Result

    The response for each call contains the detailed definition for that specific predecessor object. Repeat this process for every dependency to get the complete list of predecessors.

Notices

Trademarks

Terms and conditions for product documentation

Index