PacBio

Sequel® systems SMRT® Link web services API use cases (v11.0)

Research use only. Not for use in diagnostic procedures.

P/N 102-298-700 Version 01 (April 2022)

© 2022, PacBio. All rights reserved.

Information in this document is subject to change without notice. PacBio assumes no responsibility for any errors or omissions in this document.

PACBIO DISCLAIMS ALL WARRANTIES WITH RESPECT TO THIS DOCUMENT, EXPRESS, STATUTORY, IMPLIED OR OTHERWISE, INCLUDING, BUT NOT LIMITED TO, ANY WARRANTIES OF MERCHANTABILITY, SATISFACTORY QUALITY, NONINFRINGEMENT OR FITNESS FOR A PARTICULAR PURPOSE. IN NO EVENT SHALL PACBIO BE LIABLE, WHETHER IN CONTRACT, TORT, WARRANTY, PURSUANT TO ANY STATUTE, OR ON ANY OTHER BASIS FOR SPECIAL, CONSEQUENTIAL, INCIDENTAL, EXEMPLARY OR INDIRECT DAMAGES IN CONNECTION WITH (OR ARISING FROM) THIS DOCUMENT, WHETHER OR NOT FORESEEABLE AND WHETHER OR NOT PACBIO IS ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

Certain notices, terms, conditions and/or use restrictions may pertain to your use of PacBio products and/or third party products. Refer to the applicable PacBio terms and conditions of sale and to the applicable license terms at http://www.pacificbiosciences.com/licenses.html.

Trademarks:

Pacific Biosciences, the PacBio logo, PacBio, Circulomics, Omnione, SMRT, SMRTbell, Iso-Seq, Sequel, Nanobind, and SBB are trademarks of Pacific Biosciences of California, Inc (PacBio). All other trademarks are the sole property of their respective owners.

See https://github.com/broadinstitute/cromwell/blob/develop/LICENSE.txt for Cromwell redistribution information.

PacBio
1305 O'Brien Drive
Menlo Park, CA 94025
www.pacb.com

Sequel® Systems SMRT® Link web services API use cases (v11.0)

Introduction **3**

Connecting to the SMRT Link Services API securely **3**

How the authentication API works 4

SSL security features 6

Python example 6

How to set up a run in Run Design **8**

How to get recent runs **9**

How to monitor the progress of a SMRT Link run **10**

How to run jobs using services **12**

How to import a completed collection (Data Set) **13**

Searching for a Data Set **13**

How to capture run-level summary metrics **14**

How to get SMRT Link Data Set reports by using the UUID **14**

How to get QC reports for a specific collection **15**

How to get QC reports for a specific SMRT Link run **15**

How to set up a SMRT Link analysis job for a specific workflow **16**

How to query job history **20**

How to copy and rerun a SMRT Link analysis **21**

How to run an analysis on all collections in a run **21**

How to delete a SMRT Link job **24**

How to create and manipulate a Project **26**

How to retrieve mapping report metrics from an analysis job **27**

Introduction

The SMRT Link Web Services API, provided by PacBio, allows integration of SMRT Link with third-party software. It is also used for accessing features such as designing and performing quality control on instrument runs, querying new data from the instrument, and starting analyses on the sequence data.

This document describes common tasks performed using the SMRT Link Web Services API and provides "how to" recipes for accomplishing these tasks. To accomplish a task, you usually need to perform several API calls; the document describes the order of these calls.

As an example of a real-world workflow, most of the examples below roughly correspond to what happens internally when a Site Acceptance Test is run on Sequel II systems using SMRT Link, starting from run design and finishing with the analysis pipeline.

Note: **For clarity, all of the API examples in this document use the unauthenticated, insecure endpoints**. In a default SMRT Link installation, these are available from `localhost` on port 9091. If you are connecting from a remote host and/or you require SSL or authentication, you will instead go through the API gateway layer, which uses port 8243 and adds the prefix `/SMRTLink/1.0.0`. For example, with default installer settings, these two URLs refer to the same endpoint (assuming that the SMRT Link server is running on `localhost`):

```
http://localhost:9091/smrt-link/datasets/subreads
https://servername.serverdomain:8243/SMRTLink/1.0.0/smrt-link/datasets/subreads
```

The next section explains these connections and provides programmatic examples. For other detailed information on the SMRT Link Web Services API calls, see

`https://<SMRTLinkServername>:8243/sl/docs/services/`

where <`SMRTLinkServername`>`:8243` is the name and port number of your local SMRT Link server.

Connecting to the SMRT Link Services API securely

SMRT Link v11.0 restricts access to the services port (Default = `9091`) to clients running on `localhost` or connecting via the secure (HTTPS) API gateway on port 8243, with authentication credentials. The default API gateway is WSO2 API Manager, but v11.0 and later introduce a replacement combining `NGINX` for HTTPS, `KrakenD` for API management, and `Keycloak` for authentication. The authenticated API is used identically in either case.

If you **only** connect from `localhost`, the existing clients will continue to work as long as you specify `localhost` or `127.0.0.1` and **not** the full host/domain name. If you are running any external database or automation programs that connect to the SMRT Link API, this section describes how to adapt your code to v11.0.

Use caution when embedding user credentials in shell scripts or source code, as this may expose them in log files or shell history. We recommend that automated clients such as LIMS systems use a special-purpose account with limited or no system login access. For example, the SMRT Link v11.0 installation process automatically creates a user in WSO2 or `Keycloak` for the Sequel systems to use when connecting. Since this user is **only** known to users of SMRT Link, it **cannot** be used for any purpose other than connecting to the SMRT Link API.

How the authentication API works

Secure API access requires passing encoded authentication credentials in the HTTP header, with a slightly different endpoint URL. This is a two-step process: First the client requests an access token using the provided user name and password, then connects to the API endpoint using the access token. The token remains good for up to two hours (7200 seconds), but several caveats about token expiry are discussed below.

To connect to a secure API endpoint, follow this procedure, replacing `<servername>` with the fully-qualified domain name:

1. POST a request to `https://<servername>:8243/token` with these HTTP headers:

   ```
   Content-Type: application/x-www-form-urlencoded
   Authorization: Basic
   S01MejVnN2ZibXg4UlZGS0tkdTBOT3JKaWM0YTo2TmpSWEJjRmZMWk93SGMwWGxpZGl6NHl3Y3Nh
   ```

   and this content, replacing `<user>` and `<password>` with the actual credentials:

   ```
   {
     "grant_type": "password",
     "username": "<user>",
     "password": "<password>",
     "scope": "welcome run-design run-qc openid analysis sample-setup data-
   management userinfo"
   }
   ```

   The "Basic" authorization identifies the client to WSO2; for convenience we use hard-coded client registration credentials in SMRT Link and the `pbservice` clients, shown above. (The string passed here is a base-64 encoding of combined user and password strings.)

   The client response will look something like below: (This is an example of the WSO2 response; the `id_token` string can be ignored and is omitted here for clarity.)

   ```
   {
     'access_token': '05649edd-9b67-3754-9e5c-5347cdedbf99',
     'id_token': '<id_token>',
     'expires_in': 6272,
     'token_type': 'Bearer',
     'scope': 'analysis data-management openid run-design run-qc userinfo',
     'refresh_token': '121e02f9-3083-3a3e-b126-547e344769bd'
   }
   ```

2. Perform the API client call. The URL must now include the prefix `/SMRTLink/1.0.0`, and use HTTPS port 8243. For example, these calls are equivalent:

   ```
   GET http://localhost:9091/status
   GET https://servername:8243/SMRTLink/1.0.0/status
   ```

   Also note that **all** service endpoints that were originally prefixed with `/secondary-analysis` now need to use `/smrt-link` instead, for example:

   ```
   GET http://localhost:9091/secondary-analysis/datasets/subreads
   ```

   now becomes:

   ```
   GET https://servername:8243/SMRTLink/1.0.0/smrt-link/datasets/subreads
   ```

These HTTP headers are required, replacing `<access_token>` with the field from the response in Step 1:

```
Content-type: application/json
Authorization: Bearer <access_token>
```

3. The access token remains valid for the duration specified by `expires_in` (in seconds). In practice we find it safest to refresh sooner than this to avoid clock skew issues. You can use the refresh token to request a new `access_token` instead passing the user/password credentials:

```
{
  "grant_type": "refresh_token",
  "refresh_token": "<refresh_token>"
}
```

This is posted to the same `/token` endpoint as in Step 1, with the same headers. **Note however that if you have multiple clients running simultaneously, the refresh mechanism will effectively lead to a race condition, therefore re-authenticating each time is recommended if the clients are running for longer than the expiry time.**

4. You can revoke an access token by POSTing to `/revoke`:

```
POST https://servername:8243/revoke
```

Use the same headers as Step 1, and this body:

```
{
  "token": "<access_token>",
  "token_type_hint": "access_token"
}
```

This is what the **logout** button in SMRT Link does. (It is not, however, necessary for non-browser client applications.)

As a compact practical example, these Linux commands show how to use the secure API with the `curl` and `jq` utilities:

```
AUTH_TOKEN=$(curl -k -s --user KMLz5g7fbmx8RVFKKdu0NOrJic4a:6NjRXBcFfLZOwHc0Xlidiz4ywcsa
-d
"grant_type=password&username=$API_USER&password=$API_PASS&scope=sample-setup+run-
design+run-qc+data-management+analysis+userinfo+openid"
https://servername:8243/token | jq -r .access_token)

curl -k -s -H "Authorization: Bearer $AUTH_TOKEN"
https://servername:8243/SMRTLink/1.0.0/status
```

Here the `API_USER` and `API_PASS` variables should contain the actual user credentials; again, use caution when passing sensitive authentication information. Note that `curl` internally converts the hard-coded `--user` credentials to the appropriate basic authorization header, and also sets the `Content-Type` header automatically.

SSL security features

The full SSL/HTTPS implementation includes several checks designed to prevent "man-in-the-middle" attacks by hackers, including the reliance on central certificating authorities to sign SSL keys, which are also tied to specific host names. The default SMRT Link installation uses a generic "self-signed" certificate that can optionally be replaced with a user-provided official certificate for that site. If this is **not** done, or if you encounter other problems with SSL security features, you may need to disable these features. This does not eliminate encryption or authentication, but it is generally discouraged by HTTP client libraries and tools. For example in the shell commands shown in the previous section, the `-k` flag tells `curl` to disable SSL certificate verification.

Python example

The following source code provides a complete working example of a simple authenticated client call using only the Python 3.7 standard library plus the `request` module, equivalent to the `curl` commands above:

```
class Wso2Constants(object):
    # These client registration credentials are valid for every SMRT Link
    # server (and are also used by the SL UI)
    SECRET = "KMLz5g7fbmx8RVFKKdu0NOrJic4a"
    CONSUMER_KEY = "6NjRXBcFfLZOwHc0Xlidiz4ywcsa"
    SCOPES = ["welcome", "run-design", "run-qc", "openid", "analysis",
              "sample-setup", "data-management", "userinfo"]

def _create_auth(secret, consumer_key):
    return base64.b64encode(":".join([secret, consumer_key]).encode("utf-8"))

def _get_token(url, user, password, scopes, secret, consumer_key):
    basic_auth = _create_auth(secret, consumer_key).decode("utf-8")
    headers = {
        "Authorization": "Basic {}".format(basic_auth),
        "Content-Type": "application/x-www-form-urlencoded"
    }
    scope_str = " ".join({s for s in scopes})
    payload = dict(grant_type="password",
                   username=user,
                   password=password,
                   scope=scope_str)
    # verify is false to disable the SSL cert verification
    return requests.post(url, payload, headers=headers, verify=False)

def get_smrtlink_wso2_token(user, password, url):
    r = _get_token(url, user, password, Wso2Constants.SCOPES, Wso2Constants.SECRET,
Wso2Constants.CONSUMER_KEY)
    r.raise_for_status()
    j = r.json()
    access_token = j['access_token']
    refresh_token = j['refresh_token']
    scopes = j['scope'].split(" ")
    return access_token, refresh_token, scopes
```

```
def _to_headers(access_token):
    return {
        "Authorization": "Bearer {}".format(access_token),
        "Content-type": "application/json"
    }

def _get_endpoint(api_path, access_token):

    api_url = "https://{h}:8243/SMRTLink/1.0.0{p}".format(h=host, p=api_path)
    headers = _to_headers(access_token)

    # verify=False disables SSL verification
    response = requests.get(api_url, headers=headers, verify=False)
    response.raise_for_status()
    return response.json()

def get_status(hostname, user, password):
    token_url = "https://{h}:8243/token".format(h=host)
    access_token, refresh_token, scopes = get_smrtlink_wso2_token(user, password,
token_url)
    return _get_endpoint("/status", access_token)
```

How to set up a run in Run Design

To set up a Run Design, perform the following steps:

1. Prepare the Run Design information in an XML file. The XML file should correspond to the `PacBioDataModel.xsd` schema.
2. Create the Run Design by using the POST request with the following endpoint:

   ```
   POST /smrt-link/runs
   ```

The payload (request body) for this POST request is a JSON string with the following fields:

- `dataModel`: The serialized XML containing the Run Design information.
- `name`: The name of the run.
- `summary`: A short description of the run.

Example: Create a Run Design using the following API call:

```
POST /smrt-link/runs
```

Use the payload as in the following example:

```
{
  "dataModel" : "https://smrtlink-alpha-
nightly.nanofluidics.com:8243/sl/docs/xsd-datamodels/PacBioDataModel.xsd",
  "name" : "54001_SAT",
  "summary" : "SAT"
}
```

How to get recent runs

To get recent runs, perform the following steps:

1. Get the list of all runs by using the GET request with the following endpoint:

   ```
   GET /smrt-link/runs
   ```

2. Filter the response based on the value of the `createdAt` field. For example:

   ```
   "createdAt": "2016-12-13T19:11:54.086Z"
   ```

Note: You may also search runs based on specific criteria, such as reserved state, creator, or summary substring.

Example: Find all runs created on or after 01.01.2017

First, get the list of all runs:

```
GET /smrt-link/runs
```

The response is an array of run objects, as in the following example: (Some fields are removed for display purposes.)

```
[
    {
        "name" : "54001_SAT",
        "uniqueId" : "a836efbc-fd58-40f6-b586-43c743730fe0",
        "createdAt" : "2016-11-08T17:50:57.955Z",
        "summary" : "SAT run"
    },
    {
        "name" : "54001_ecoli_15k",
        "uniqueId" : "798ff161-23ee-433a-bfd9-be8361b40f15",
        "createdAt" : "2017-01-20T16:08:41.610Z",
        "summary" : "E. coli assembly"
    },
    {
        "name" : "54001_hla_amplicons",
        "uniqueId : "5026afad-fbfa-407a-924b-f89dd019ca9f",
        "createdAt" : "2017-01-21T00:21:52.534Z",
        "summary" : "Human HLA"
    }
]
```

Now, search the above response for all run objects whose `createdAt` field starts with the `2017-01` substring. In the above example, you will get two runs that fit your criteria (that is, created on or after 01.01.2017):

- Run with "name" equal to "54001_ecoli_15k",
- Run with "name" equal to "54001_hla_amplicons".

How to monitor the progress of a SMRT Link run

Run progress can be monitored by looking at the completion status of each Collection associated with that run. Perform the following steps:

1. If you do not have the Run UUID, retrieve it as described in Step 1 of How to get recent runs.

2. Once you have the Run UUID, get all Collections that belong to the run. Use the Run UUID in the GET request with the following endpoint:

   `GET /smrt-link/runs/{runUUID}/collections`

   The response contains the list of all Collections of that run.

3. Monitor Collection status to see when all Collections are complete.

   Until all Collections of the run have the field `status` set to `Complete`, repeat the GET request with the following endpoint:

   `GET /smrt-link/runs/{runUUID}/collections`

   You may also monitor each Collection individually.

   Use the Collection UUID in the GET request with the following endpoint:

   `GET /smrt-link/runs/{runUUID}/collections/{collectionUUID}`

4. To monitor run progress using QC metrics as well, do this at the Collection level, for each Collection that belongs to this run. For instructions, see How to get QC reports for a specific collection.

The full set of QC metrics for a Collection will be available **only** when the Collection is **complete**. Monitor the completion status of each Collection and, for each complete Collection, check its QC metrics. QC metrics of all Collections that belong to the run will let you evaluate the overall success of the run.

Example

To monitor the run with Name = `54001_DryRun_2Cells_20161219`, use the following steps:

1. Get the list of all runs as described in the previous section.

   `GET /smrt-link/runs`

   The response is an array of run objects, as in the following example: (Some fields are removed for display purposes.)


```
[
    {
        "name" : "54001_SAT",
        "uniqueId" : "a836efbc-fd58-40f6-b586-43c743730fe0",
        "createdAt" : "2016-11-08T17:50:57.955Z",
        "summary" : "SAT run"
    },
    {
        "name" : "54001_ecoli_15k",
        "uniqueId" : "798ff161-23ee-433a-bfd9-be8361b40f15",
        "createdAt" : "2017-01-20T16:08:41.610Z",
        "summary" : "E. coli assembly"
    },
    {
```

```
        "name" : "54001_hla_amplicons",

        "uniqueId" : "5026afad-fbfa-407a-924b-f89dd019ca9f",
        "createdAt" : "2017-01-21T00:21:52.534Z",
        "summary" : "Human HLA"
    }
]
```


2. Search the above response for the object with the `name` field equal to `54001_SAT`.

   From the above example, you will get the run object with the `uniqueId` field equal to `a836efbc-fd58-40f6-b586-43c743730fe0`.

3. With this Run UUID = `a836efbc-fd58-40f6-b586-43c743730fe0`, get all Collections that belong to this run:

   ```
   GET /smrt-link/runs/a836efbc-fd58-40f6-b586-43c743730fe0/collections
   ```

   The response is an array of Collection objects of this run, as in the following example:


```
[{
    "name" : "54001_SAT_1stCell",
    "instrumentName" : "Sequel",
    "context" : "m54001_161219_161247",
    "well" : "A01",
    "status" : "Complete",
    "instrumentId" : "54001",
    "startedAt" : "2016-12-19T16:12:47.014Z",
    "uniqueId" : "7cf74b62-c6b8-431d-b8ae-7e28cfd8343b",
    "collectionPathUri" : "/data/sequel/r54001_20161219_160902/1_A01",
    "runId" : "a836efbc-fd58-40f6-b586-43c743730fe0",
    "movieMinutes" : 120
}, {
    "name" : "54001_SAT_2ndCell",
    "instrumentName" : "Sequel",
    "context" : "m54001_161219_184813",
    "well" : "B01",
    "status" : "Ready",
    "instrumentId" : "54001",
    "startedAt" : "2016-12-19T16:12:47.014Z",
    "uniqueId" : "08af5ab4-7cf4-4d13-9bcb-ae977d493f04",
    "collectionPathUri" : "/data/sequel/r54001_20161219_160902/2_B01",
    "runId" : "a836efbc-fd58-40f6-b586-43c743730fe0",
    "movieMinutes" : 120
}
]
```


   In the above example, the first Collection has `status` of `Complete`.

   You can take its UUID, i.e. `uniqueId: 7cf74b62-c6b8-431d-b8ae-7e28cfd8343b`, and get its QC metrics. For instructions, see How to get QC reports for a specific collection.

   The second Collection has a `status` of `Ready`.

   You can take its UUID, i.e. `uniqueId: 08af5ab4-7cf4-4d13-9bcb-ae977d493f04`, and monitor its status until it becomes `Complete`. To do so, use the following API call:

   ```
   GET /smrt-link/runs/a836efbc-fd58-40f6-b586-
   43c743730fe0/collections/08af5ab4-7cf4-4d13-9bcb-ae977d493f04
   ```

Once this Collection becomes complete, you can get its QC metrics as well.

How to run jobs using services

SMRT Link runs several different types of "jobs" which consist of tasks that may take an arbitrarily long time to run and are therefore executed asynchronously. To view a list of supported job types, enter:

```
GET /smrt-link/job-manager/job-types
```


```
  [
    {
      "jobTypeId": "db-backup",
      "description": "Create a DB backup of the SMRT Link system",
      "isQuick": true,
      "isMultiJob": false
    },
    {
      "jobTypeId": "delete-datasets",
      "description": "(Soft) delete of PacBio DataSet XML",
      "isQuick": true,
      "isMultiJob": false
    },
    ...
  ]
```


Note: "Quick" jobs (generally taking less than a minute) have their own queue, separate from analysis jobs and other I/O intensive tasks.

Creating a job follows this pattern:

```
POST /smrt-link/job-manager/jobs/<jobTypeId>
```

The request body varies depending on job type, from a single path field to more complex data types, several examples of which are described below. The server should respond with `201: Created` and the model for the new job:


```
  {
    "name": "import-dataset",
    "updatedAt": "2018-06-19T21:13:31.047Z",
    "workflow": "{}",
    "path": "/smrtlink/userdata/jobs_root/000/000001",
    "state": "CREATED",
    "tags": "",
    "uuid": "7cf74b62-c6b8-431d-b8ae-7e28cfd8343b",
    "projectId": 1,
    "jobTypeId": "import-dataset",
    "id": 1,
    "smrtLinkVersion": "6.0.0.SNAPSHOT38748",
    "comment": "Description for job Import PacBio DataSet",
    "createdAt": "2018-06-19T21:13:31.047Z",
    "isActive": true,
    "createdBy": null,
    "isMultiJob": false,
    "jsonSettings":
"{\"path\":\"/data/sequel/r54001_20161219_160902/1_A01/m54001_20161219_170101.

subreadset.xml\",\"datasetType\":\"PacBio.DataSet.SubreadSet\",\"submit\":true}",

    "jobUpdatedAt": "2018-06-19T21:13:31.047Z",
  }
```

Client code should now block until the job is complete, which should result in the `state` field changing to `SUCCESSFUL` if all goes well.

Note: Blocking mean that the client will poll for the server to complete a Job. When the system is under minimal load, blocking can be used instead of manually polling for the job to complete. High-computational situations, such as a large FASTA file conversion, are **not** appropriate for blocking.

How to import a completed collection (Data Set)

Once a run is complete and the data have been transfered off the instrument, the resulting Data Set(s) can be imported into SMRT Link. This creates an `import-dataset` job that runs asynchronously and generates several reports used to assess run quality.

To import a Data Set, use this API call:

```
POST /smrt-link/job-manager/jobs/import-dataset
```

The request body in this case is very simple:

```
 {
  "path":
"/data/sequel/r54001_20161219_160902/1_A01/m54001_20161219_170101.subreadset.xml"
 }
```

The server should respond with `201: Created` and the model for the new job; it should only take several minutes at most for the import job to complete.

Note that the same `import-dataset` job type is also used to import other Data Set types such as the ReferenceSet XML used to run the SAT pipeline.

Searching for a Data Set

The Data Set retrieval endpoints support a number of search operators that may be included as CGI parameters:

```
GET /smrt-link/datasets/subreads?name=human
```

String fields use case-insensitive partial matching, so this will retrieve all Data Sets whose names include `human` in any combination of upper and lower case.

You can also retrieve a selection of Data Sets by posting a search query with a list of UUIDs:

```
POST /smrt-link/datasets/subreads
  {
    "uuid": "in:7cf74b62-c6b8-431d-b8ae-7e28cfd8343b,a836efbc-fd58-40f6-b586-
43c743730fe0"
  }
```

Note: The list needs to start with `in:` to tell the search API to find values from a list.

How to capture run-level summary metrics

Run-level summary metrics are captured in the QC reports. See the following sections:

- How to get QC reports for a specific SMRT Link run.
- How to get QC reports for a specific collection.

How to get SMRT Link Data Set reports by using the UUID

To get reports for a Data Set, given the Data Set UUID, perform the following steps:

1. Determine the Data Set type from the list of available types. Use the GET request with the following endpoint:

   ```
   GET /smrt-link/dataset-types
   ```

2. Get the corresponding Data Set type string. The Data Set type is in the `shortName` field.

3. Get reports that correspond to the Data Set. Given the Data Set UUID and the Data Set type, use them in the GET request with the following endpoint:

   ```
   GET /smrt-link/datasets/{datasetType}/{datasetUUID}/reports
   ```

Example:

To get reports associated with a subreadset with UUID = 146338e0-7ec2-4d2d-b938-11bce71b7ed1, perform the following steps:

Use the GET request with the following endpoint:

```
GET /smrt-link/dataset-types
```

You see that the `shortName` of SubreadSets is `subreads`. The desired endpoint is:

```
/smrt-link/datasets/subreads/7cf74b62-c6b8-431d-b8ae-7e28cfd8343b/reports
```

Use the GET request with this endpoint to get reports that correspond to the SubreadSet with UUID = 7cf74b62-c6b8-431d-b8ae-7e28cfd8343b:

```
GET /smrt-link/datasets/subreads/7cf74b62-c6b8-431d-b8ae-7e28cfd8343b/reports
```

Once you have the UUID for an individual report, download it using the datastore files service with the `uuid` field:

```
GET /smrt-link/datastore-files/519817b6-4bfe-4402-a54e-c16b29eb06eb/download
```

How to get QC reports for a specific collection

For completed Collections, the Collection UUID is the same as the UUID of the SubreadSet for that Collection. To retrieve the QC reports of a completed Collection, given the Collection UUID, perform the following steps:

1. Get the QC reports that correspond to this Collection by using the GET request with the following endpoint:

   `GET /smrt-link/datasets/subreads/{collectionUUID}/reports`

   See How to get SMRT Link reports for Data Sets by using the UUID for more details.

Note: Obtaining Data Set reports based on the Collection UUID as described above will only work if the Collection is **complete**. If the Collection is **not** complete, then the SubreadSet does not exist yet.

How to get QC reports for a specific SMRT Link run

To get QC reports for a specific run, given the run Name, perform the following steps:

1. Get the list of all runs by using the GET request with the following endpoint:

   `GET /smrt-link/runs`

   In the response, perform a text search for the run name: Find the object whose `name` field is equal to the run name, and get the Run UUID, which is found in the `uniqueId` field.

2. Get all Collections that belong to this run by using the Run UUID found in the previous step in the GET request with the following endpoint:

   `GET /smrt-link/runs/{runUUID}/collections`

3. Take a Collection UUID of one of Collection objects received in the previous response. The Collection UUIDs are in the `uniqueId` fields.

   For **complete** Collections, the Collection UUID is the same as the UUID of the SubreadSet for that Collection.

   Make sure that the Collection whose `uniqueId` field you take has the field `status` set to `Complete`. This is because obtaining Data Set reports based on the Collection UUID as described below will **only** work if the Collection is **complete**. If the Collection is **not** complete, the SubreadSet does not exist yet.

   You can now retrieve the QC reports that correspond to this Collection as described in How to get SMRT Link reports for Data Sets by using the UUID.

4. Repeat Step 3 to download QC reports for all complete Collections of that run.

Example

You view the Run QC page in SMRT Link, and open the page of a run with a status of **Complete**. Take the run name and look for the Run UUID in the list of all runs, as described above.

Note: The Run ID also appears in the `{runUUID}` path parameter of the SMRT Link UI URL:

`http://SMRTLinkServername.domain:9090/#/run-qc/{runUUID}`

So the shorter way would be to take the Run UUID **directly** from the URL, such as

`http://SMRTLinkServername.domain:9090/#/run-qc/a836efbc-fd58-40f6-b586-43c743730fe0`

With this Run UUID = a836efbc-fd58-40f6-b586-43c743730fe0, get all Collections that belong to this run:

`GET /smrt-link/runs/a836efbc-fd58-40f6-b586-43c743730fe0/collections`

Take a UUID of a completed Collection, such as `uniqueId: 59230aeb-a8e3-4b46-b1b1-24c782c158c1`. With this Collection UUID, retrieve QC reports of the corresponding SubreadSet:

`GET /smrt-link/datasets/subreads/7cf74b62-c6b8-431d-b8ae-7e28cfd8343b/reports`

Take a UUID of some report, such as `uuid: 00c310ab-e989-4978-961e-c673b9a2b027`. With this report UUID, download the corresponding report file:

`GET /smrt-link/datastore-files/00c310ab-e989-4978-961e-c673b9a2b027/download`

Repeat the last two API calls until you download all desired reports for all complete Collections.

How to set up a SMRT Link analysis job for a specific workflow

Note to users of SMRT Link v7.0.0 or earlier: The analysis engine (`pbsmrtpipe`) was replaced in SMRT Link v8.0 with `Cromwell`, developed by the Broad Institute, and the names of workflows and options have changed significantly. The data model remains the same, as does most of the services API (aside from the change of job type ID). See **SMRT Tools reference guide (v11.0)** for details about `Cromwell` changes. Note that for backwards compatibility, the terms "pipeline" and "workflow" are used interchangeably when referring to specific applications.

To create an analysis job for a specific workflow, you need to create a job of type `analysis` with the payload based on the template of the desired pipeline. Perform the following steps:

1. Get the list of all pipeline templates used for creating analysis jobs:

   `GET /smrt-link/resolved-pipeline-templates`

2. In the response, search for the name of the specific pipeline to set up. Once the desired template is found, note the values of the pipeline `id` and `entryPoints` elements of that template.

3. Identify the Data Set(s) you want to use to run the analysis, and make note of the UUID(s).

4. For each entry point, find the corresponding record in the `dataset-types` endpoint, and extract the `shortName` field:

   `GET /smrt-link/dataset-types`

5. For each input Data Set, check whether a record already exists at the appropriate Data Set endpoint, and if one does not, it should be imported as described above. The Data Set endpoints take this form:

   `GET /smrt-link/datasets/<shortName>/UUID`

6. Build the request body for creating a job of type `analysis`. The basic structure looks like this:

```
{
    "entryPoints": [
        {
            "datasetId": "5bd43ef4-6afe-dc62-4f49-03b75a051801",
            "entryId": "eid_subread",
            "fileTypeId": "PacBio.DataSet.SubreadSet"
        },
```

```
            {
                "datasetId": "1a369917-507e-4f70-9f38-69614ff828b6",
                "entryId": "eid_ref_dataset",
                "fileTypeId": "PacBio.DataSet.ReferenceSet"
            }
        ],
        "name": "Lambda SAT job",
        "pipelineId": "cromwell.workflows.pb_sat",

        "taskOptions": [],
        "workflowOptions": []
    }
```

Use the pipeline `id` found in Step 2 as the value for the `pipelineId` element.

Use Data Set types of the `entryPoints` array found in Step 1 and corresponding Data Set IDs found in Step 2 as the values for elements of the `entryPoints` array.

The Data Set IDs may be provided **either** as UUIDs (which are specified by the XML file and are independent of the server used) or integer IDs (which are generated by the server when the Data Sets are imported). In most cases the UUIDs will be easier to work with as they are known in advance.

Note that the `taskOptions` array is **optional** and may be completely empty in the request body. (`workflowOptions` is not only optional but the contents are ignored by the server.)

7. Create a job of type `analysis`. Use the request body built in the previous step in the POST request with the following endpoint:

   ```
   POST /smrt-link/job-manager/jobs/analysis
   ```

8. You may monitor the state of the job created in Step 6 with the following request:

   ```
   GET /smrt-link/job-manager/jobs/analysis/{jobID}/events
   ```

   where jobID is equal to the value received in the `id` element of the response in Step 6.

Example

Suppose you want to set up an analysis job for the SAT pipeline.

First, get the list of all pipeline templates used for creating analysis jobs:

```
GET /smrt-link/resolved-pipeline-templates
```

The response is an array of pipeline template objects. In this response, do the search for the entry with `name : Site Acceptance Test (SAT)`. The entry may look as in the following example: (Task options were truncated for clarity.)

```
    {
        "name": "Site Acceptance Test (SAT)",
        "id" : "cromwell.workflows.pb_sat",
        "description": "Site Acceptance Test - lambda genome resequencing used
to validate new\n    PacBio installations",
        "version" : "0.1.0",
        "entryPoints": [
            {
                "entryId": "eid_ref_dataset",
                "fileTypeId": "PacBio.DataSet.ReferenceSet",
```

```
                "name": "Entry Name: PacBio.DataSet.ReferenceSet"
            },
            {
                "entryId": "eid_subread",
                "fileTypeId": "PacBio.DataSet.SubreadSet",
                "name": "Entry Name: PacBio.DataSet.SubreadSet"
            }
        ],
        "tags" : [ "consensus", "mapping", "reports", "sat"],
        "taskOptions" : [{
            {
                "default": "",
                "description": "Dataset filter string",
                "id": "dataset_filters",
                "name": "Filters to add to the DataSet",
                "optionTypeId": "string"
            },
            {
                "default": 0,
                "description": "Downsampling Factor",
                "id": "downsample_factor",
                "name": "Downsampling Factor",
                "optionTypeId": "integer"
            }
        ]    }
```

In the above entry, take the value of the pipeline `id :cromwell.workflows`.

Also, take the Data Set types of `entryPoints` elements: `fileTypeId : PacBio.DataSet.SubreadSet` and `fileTypeId : PacBio.DataSet.ReferenceSet`. In this example we use the lambdaNEB reference and example PacBio data that are distributed with SMRT Link. First check whether they have been imported already:

```
GET /smrt-link/datasets/subreads/5bd43ef4-6afe-dc62-4f49-03b75a051801
```

```
  {
    "name": "lambda/0007_tiny",
    "updatedAt": "2015-10-26T22:54:46.000Z",
    "path": "opt/smrtlink-release_6.0.0.40259/admin/bin/../../bundles/smrtinub
/current/private/pacbio/canneddata/lambdaTINY/m150404_101626_42267_c1008079208
000000018231741102915l4_s1_p0.subreadset.xml",
    "instrumentControlVersion": "2.3.0.1.142990",
    "tags": "",
    "instrumentName": "42267",
    "uuid": "5bd43ef4-6afe-dc62-4f49-03b75a051801",
    "totalLength": 16865720,
    "projectId": 1,
    "numRecords": 19930,
    "wellSampleName": "Inst42267-040315-SAT-100pM-2kb-P6C4",
    "bioSampleName": "unknown",
    "version": "3.0.1",
    "cellId": "unknown",
    "id": 5,
    "md5": "288d3bdadf83bda41dd7fefc11cad128",
    "importedAt": "2018-07-06T00:45:10.753Z",
    "jobId": 3,
    "createdAt": "2015-10-26T22:54:46.000Z",
    "isActive": true,
```

```
    "createdBy": "smrtlinktest",

    "wellName": "A01",
    "cellIndex": 4,
    "metadataContextId":
"m150404_101626_42267_c100807920800000001823174110291514_s1_p0",

    "numChildren": 0,
    "runName": "lambdaTINY",
    "datasetType": "PacBio.DataSet.SubreadSet",
    "comments": "Inst42267-SAT-100pM-2kbLambda-P6C4-Std120_CPS_040315"

  }
GET /smrt-link/datasets/references/1a369917-507e-4f70-9f38-69614ff828b6
  {
    "name": "lambdaNEB",
    "updatedAt": "2015-10-24T03:32:50.530Z",
    "path": "opt/smrtlink-release_6.0.0.40259/admin/bin/../../bundles/smrtinub
/current/private/pacbio/canneddata/referenceset/lambdaNEB/referenceset.xml",
    "ploidy": "haploid",
    "tags": "",
    "uuid": "1a369917-507e-4f70-9f38-69614ff828b6",
    "totalLength": 48502,
    "projectId": 1,
    "numRecords": 1,
    "version": "3.0.1",
    "id": 4,
    "md5": "4861bca63e02aa26c92724febb3299c2",
    "importedAt": "2018-07-06T00:45:10.660Z",
    "jobId": 5,
    "createdAt": "2015-10-24T03:32:50.530Z",
    "isActive": true,
    "createdBy": "smrtlinktest",
    "organism": "lambdaNEB",
    "numChildren": 0,
    "datasetType": "PacBio.DataSet.ReferenceSet",
    "comments": "reference dataset comments"
  }
```


Build the request body for creating an `analysis` job for the SAT pipeline. Use the pipeline `id` obtained above as the value for the `pipelineId` element. Use the two Data Set UUIDs as values of the `datasetId` fields in the `entryPoints` array. For example:


```
{
        "pipelineId" : "cromwell.workflows.pb_sat",
        "entryPoints" : [
            {
                "datasetId": "5bd43ef4-6afe-dc62-4f49-03b75a051801",
                "entryId": "eid_subread",
                "fileTypeId": "PacBio.DataSet.SubreadSet"
            },
            {
                "datasetId": "1a369917-507e-4f70-9f38-69614ff828b6",
                "entryId": "eid_ref_dataset",
                "fileTypeId": "PacBio.DataSet.ReferenceSet"
            }
        ],
        "taskOptions" : [],
```

```
        "workflowOptions": [],
        "name": "My SAT Job"
    }
```

Now create a job of type `analysis`. Use the request body built above in the following API call:

```
POST /smrt-link/job-manager/jobs/analysis
```

Verify that the job was created successfully. The return HTTP status should be `201 Created`.

How to query job history

The Job Service endpoints provide a number of search criteria (plus paging support) that can be used to limit the return results. A full list of available search criteria is provided in the the JSON Swagger API definition for the jobs endpoint. The following search retrieves all **failed** Site Acceptance Test (SAT) pipeline jobs:

```
GET /smrt-link/job-manager/jobs/analysis?state=FAILED&subJobTypeId=cromwell
```

For most data types, additional operators besides equality are allowed. For example:

```
GET /smrt-link/job-manager/jobs/analysis?createdAt=lt%3A2019-03-
01T00:00:00.000Z&createdBy=myusername
```

This retrieves all `analysis` jobs run before 2019-03-01 by a user with the login ID `myusername`.

Note: Certain searches, especially partial text searches using `like:`, may be significantly slower to execute and can overload the server if performed too frequently.

You can also perform bulk retrieval of jobs using the search endpoint:

```
POST /smrt-link/job-manager/jobs/analysis/search
  {
    "id": "in:1,2,3,4"
  }
```

The example above will retrieve jobs 1-4. You may also query on UUID or any other supported search field.

How to copy and rerun a SMRT Link analysis

The options endpoint for a specific job provides the POST content that ran it:

```
GET /smrt-link/job-manager/jobs/analysis/<jobId>/options
```

As is the case for Data Set IDs, **either** the UUID or the integer ID of the job can be provided. In this case, as both are generated automatically at job creation time, there is no preference for one or the other.

For example:

```
GET /smrt-link/job-manager/jobs/analysis/3/options

  {
    "name": "sat_lambda",
    "entryPoints": [
      {
        "entryId": "eid_subread",
        "fileTypeId": "PacBio.DataSet.SubreadSet",
        "datasetId": 1
      },
      {
        "entryId": "eid_ref_dataset",
        "fileTypeId": "PacBio.DataSet.ReferenceSet",
        "datasetId": 2
      }
    ],
    "workflowOptions": [],
    "taskOptions": [],

    "pipelineId": "cromwell.workflows.pb_sat"
  }
```

This data model can be directly POSTed to the `analysis` job endpoint as described above. Note that in this case, the `datasetId` fields are the integer IDs generated by the SMRT Link database backend. You can retrieve the full Data Set records (including their UUIDs) by using the same Data Set endpoints described previously, only with the integer IDs instead of UUIDs:

```
GET /smrt-link/datasets/subreads/1
GET /smrt-link/datasets/references/2
```

How to run an analysis on all collections in a run

As explained earlier, each Collection corresponds to a SubreadSet Data Set. To run an analysis on multiple SubreadSets combined, you can **either** first run a merge job to generate a single input, or let the `analysis` job perform the merge automatically.

For the two-step approach, perform the following steps:

1. As described previously, collect the UUIDs for the Collections in the Run you want to analyse.
2. Check each Collection UUID to make sure the SubreadSet XML has already been imported, and if not, import it as described above:

   ```
   GET /smrt-link/datasets/subreads/<UUID>
   ```

3. Build a payload using the following model:

```
    {
      "datasetType": "PacBio.DataSet.SubreadSet",
      "ids": ["<UUID1>", "<UUID2>", ...],
      "name": "Merge run <runId> collections"
    }
```

4. Create a `merge-datasets` job with the request body from Step 3:

```
POST /smrt-link/job-manager/jobs/merge-datasets
```

5. Block until this job completes successfully, then retrieve the list of job datastore files. One of these should be the merged Data Set.

```
GET /smrt-link/job-manager/jobs/merge-datasets/<ID>/datastore
```


```
  [
    {
      "modifiedAt": "2018-07-12T21:38:34.815Z",
      "name": "Auto-merged hdfsubreads @ 1531431514119",
      "fileTypeId": "PacBio.DataSet.SubreadSet",
      "path":
"/opt/smrtlink_5.1.0.14963/userdata/jobs_root/008/008767/merged.dataset.xml",
      "description": "Merged PacBio DataSet from 4 files",
      "uuid": "f54694da-5985-42b9-9a9e-f2190bd3b4a4",
      "fileSize": 33495,
      "importedAt": "2018-07-12T21:38:35.085Z",
      "jobId": 4,
      "createdAt": "2018-07-12T21:38:34.815Z",
      "isActive": true,
      "jobUUID": "127619b4-f615-4c3f-b208-e1bf52bfe21b",
      "sourceId": "pbscala::merge_dataset"

    },
    {
      "modifiedAt": "2018-07-12T21:38:34.264Z",
      "name": "SMRT Link Job Log",
      "fileTypeId": "PacBio.FileTypes.log",
      "path":
"/opt/smrtlink_5.1.0.14963/userdata/jobs_root/008/008767/pbscala-job.stdout",
      "description": "SMRT Link Job Log",
      "uuid": "b19fbfc6-0808-40fc-917b-092f369180cd",
      "fileSize": 388,
      "importedAt": "2018-07-12T21:38:34.266Z",
      "jobId": 8767,

      "createdAt": "2018-07-12T21:38:34.264Z",
      "isActive": true,
      "jobUUID": "127619b4-f615-4c3f-b208-e1bf52bfe21b",
      "sourceId": "analysis::master.log"
    }
  ]
```


6. You may now follow the steps for running an analysis job, using the new merged SubreadSet as input.

To use the auto-merge capability, just submit the `analysis` job options with a separate `eid_subread` entry point for each input Data Set, for example:

```
GET /smrt-link/job-manager/jobs/analysis/3/options
{
  "name": "sat_lambda",
  "entryPoints": [
    {
      "entryId": "eid_subread",
      "fileTypeId": "PacBio.DataSet.SubreadSet",
      "datasetId": "<UUID1>"
    },
    {
      "entryId": "eid_subread",
      "fileTypeId": "PacBio.DataSet.SubreadSet",
      "datasetId": "<UUID2>"
    },
    {
      "entryId": "eid_ref_dataset",
      "fileTypeId": "PacBio.DataSet.ReferenceSet",
      "datasetId": "<REF_UUID>"
    }
  ],
  "workflowOptions": [],
  "taskOptions": [],
  "pipelineId": "cromwell.workflows.pb_sat"
}
```

Note that this process is opaque to `Cromwell`, which does **not** itself accept multiple inputs with the same identifier.

How to delete a SMRT Link job

To delete a job, you need to create another job of type `delete-job`, and pass the UUID of the job to delete in the payload (the request body).

Perform the following steps:

1. Build the payload for the POST request as a JSON with the following fields:

   - `jobId`: The UUID of the job to be deleted.
   - `removeFiles`: A boolean flag specifying whether to remove files associated with the job being deleted.
   - `dryRun`: A boolean flag to check whether it is safe to delete the job prior to actually deleting it.

   Note: To make sure that it is safe to delete the job (that is, there is no other piece of data dependent on the job being deleted), then first set the `dryRun` field to `true` and perform the API call described in Step 2 below. If the call succeeds, meaning that the job can be safely deleted, set the `dryRun` field to `false` and repeat the same API call again, as described in Step 3 below.

2. Check whether the job can be deleted, without actually changing anything in the database or on disk. Create a job of type `delete-job` with the payload which has `dryRun = true`; use the POST request with the following endpoint:

   ```
   POST /smrt-link/job-manager/jobs/delete-job
   ```

3. If the previous API call succeeded, that is, the job may be safely deleted, then proceed with actually deleting the job.

   Create a job of type `delete-job` with the payload which has `dryRun = false`; use the POST request with the following endpoint:

   ```
   POST /smrt-link/job-manager/jobs/delete-job
   ```

Suppose you want to delete the job with UUID = 13957a79-1bbb-44ea-83f3-6c0595bf0d42. Define the payload as in the following example, and set the `dryRun` field to `true`:

```
{
    "jobId" : "13957a79-1bbb-44ea-83f3-6c0595bf0d42",
    "removeFiles" :true,
    "dryRun" : true
}
```

Create a job of type `delete-job`, using the above payload in the following POST request:

```
POST /smrt-link/job-manager/jobs/delete-job
```

Verify that the response status is `201: Created`.

Also notice that the response body contains JSON corresponding to the job to be deleted, as in the following example:

```
{
    "name" : "Job merge-datasets",
    "uuid" : "13957a79-1bbb-44ea-83f3-6c0595bf0d42",
    "jobTypeId" : "merge-datasets",
    "id" : 53,
    "createdAt" : "2016-01-29T00:09:58.462Z",
```

```
        ...
        "comment" : "Merging Datasets
MergeDataSetOptions(PacBio.DataSet.SubreadSet, Auto-merged subreads
@1454026198403)"
    }
```

Define the payload as in the following example, and this time set the `dryRun` field to `false`, to actually delete the job:

```
    {
        "jobId" : "13957a79-1bbb-44ea-83f3-6c0595bf0d42",
        "removeFiles" : true,
        "dryRun" : false
    }
```

Create a job of type `delete-job`, using the above payload in the following POST request:

```
POST /smrt-link/job-manager/jobs/delete-job
```

Verify that the response status is `201: Created`. Notice that this time the response body contains JSON corresponding to the job of type `delete-job`, as in the following example:

```
    {
        "name" : "Job delete-job",
        "uuid" : "1f60c976-e426-43b5-8ced-f8139de6ceff",
        "jobTypeId" : "delete-job",
        "id" : 7666,
        "createdAt" : "2017-03-09T11:51:38.828-08:00",
        ...
        "comment" : "Deleting job 13957a79-1bbb-44ea-83f3-6c0595bf0d42"
    }
```

Clients should then block until the job is complete.

How to create and manipulate a Project

By default, **all** Data Sets and analyses are part of a "General Project" with global permissions. Creating new projects lets you organize related Data Sets and jobs and optionally restrict access to specific users using the SMRT Link UI. (**Note**:This is **only** enforced in the UI itself; the REST services do **not** currently restrict users to specific projects.)

The Projects service requires user credentials, which typically means going through the WSO2 server as described in How the authentication API works. Following is an example of how to create a project that contains two Data Sets and includes three users with varying levels of access:

```
POST /SMRTLink/1.0.0/smrt-link/projects
{
    "name": "Human Structural Variation",
    "description": "Human SV datasets and analyses",
    "state": "CREATED",
    "datasets": [
        {"id": 34},
        {"id": 45}
    ],
    "members": [
        {"login": "user1", "role": "OWNER"},
        {"login": "user2", "role": "CAN_EDIT"},
        {"login": "collaborator1", "role": "CAN_VIEW"}
    ],
}
```

The server will return the newly-created project including the integer ID that should be used in subsequent requests:

```
{
    "id": 2,
    "name": "Human Structural Variation",
    "description": "Human SV datasets and analyses",
    "state": "CREATED",
    "createdAt": "2020-06-01T11:51:38.828-08:00",
    "updatedAt": "2020-06-01T11:51:38.828-08:00",
    "isActive": true,
    "grantRoleToAll": null,
    "datasets": [
        {"id": 34},
        {"id": 45}
    ],
    "members": [
        {"login": "user1", "role": "OWNER"},
        {"login": "user2", "role": "CAN_EDIT"},
        {"login": "collaborator1", "role": "CAN_VIEW"}
    ]
}
```

If you do not want to manage user permissions individually, the field `grantRoleToAll` grants global access to the project if non-null.

You can retrieve details for a specific project by appending the integer ID to the URL, thus:

```
GET /SMRTLink/1.0.0/smrt-link/projects/2
{
    "id": 2,
    "name": "Human Structural Variation",
    ...
}
```

To update project details, send a PUT request to this same URL with the modified version of the data model used to create the project. Use DELETE to soft-delete the project and reassign all of its Data Sets and jobs to the “General Project”.

Once you have a project ready to work with, any new job can be added to that project by overriding the `projectId` field to the job data model (the default is `1`, the "General Project"). You may also add a `projectId` query argument when retrieving lists of jobs or Data Sets to filter the list to members of the specified project.

How to retrieve mapping report metrics from an analysis job

The jobs API provides an endpoint to retrieve report files:

```
GET /smrt-link/job-manager/jobs/analysis/1001/reports

[
  {
    "dataStoreFile": {
      "modifiedAt": "2020-12-07T10:14:03.121Z",
      "name": "Report mapping_stats_ccs",
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "path": "/data/smrtlink/jobs-root/cromwell-
executions/pb_align_ccs/00eae20b-f029-4a64-b882-8d95038ee89e/call-
ccs_mapping/mapping/fc30ff7d-de59-49d9-800d-dd25bf749704/call-
mapping_stats/execution/mapping_stats.report.json",
      "description": "PacBio Report mapping_stats_ccs (a7d394be-9a9a-457a-
b0ad-5d41a21a460b)",
      "uuid": "a7d394be-9a9a-457a-b0ad-5d41a21a460b",
      "fileSize": 7459,
      "importedAt": "2020-12-07T10:14:46.635Z",
      "createdAt": "2020-12-07T10:14:03.121Z",
      "isActive": true,
      "sourceId": "pb_align_ccs.report_mapping_stats"
    },
    "reportTypeId": "pb_align_ccs.report_mapping_stats"
  },
  {
    "dataStoreFile": {
      "modifiedAt": "2020-12-07T10:14:12.639Z",
      "name": "Report coverage",
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "path": "/data/smrtlink/jobs-root/cromwell-
executions/pb_align_ccs/00eae20b-f029-4a64-b882-8d95038ee89e/call-
ccs_mapping/mapping/fc30ff7d-de59-49d9-800d-dd25bf749704/call-

coverage_reports/coverage_reports/408518ac-80e5-4f48-9185-ed1b51fe43c6/call-
pbreports_coverage/execution/coverage.report.json",
      "description": "PacBio Report coverage (52adc5b2-b1b9-4e27-9ae4-
94de2a525b1e)",
      "uuid": "52adc5b2-b1b9-4e27-9ae4-94de2a525b1e",
```

```
        "fileSize": 2596,
        "importedAt": "2020-12-07T10:14:46.635Z",
        "createdAt": "2020-12-07T10:14:12.639Z",
        "isActive": true,
        "sourceId": "pb_align_ccs.report_coverage"
      },
      "reportTypeId": "pb_align_ccs.report_coverage"
    }
  ]
```

For workflows that produce a mapping report, it will typically have a `reportTypeId` of the form `workflow_id.report_mapping_stats`, in this case `pb_align_ccs.report_mapping_stats`. We can then retrieve the full report by adding the UUID to the URL:

```
  GET /smrt-link/job-manager/jobs/analysis/1001/reports/a7d394be-9a9a-457a-
b0ad-5d41a21a460b

  {
    "version": "1.0.1",
    "id": "mapping_stats_ccs",
    "_comment": "Generated with pbcommand version  at 2020-12-
07T02:14:03.119926",
    "title": "Report mapping_stats_ccs",
    "attributes": [
      {
        "id": "mapping_stats_ccs.blast_identity",
        "name": "Mean Concordance (mapped)",
        "value": 0.8919674526217451
      },
      ...
    ]
  }
```

In SMRT Link v11.0 and later `blast_identity` is the metric displayed as **Mean Concordance (mapped)** in the SMRT Link UI (after converting to a percentage value). This Python snippet shows an alternative approach to retrieving several metrics like this from the report JSON file, bypassing the final API call:

```
.. code-block:: python

  import os.path
  from pbcommand.pb_io import load_report_from_json

  def get_mapping_metrics(report_file):
      report = load_report_from_json(report_file)
      keys = {"blast_identity", "mapped_reads_n", "mapped_readlength_mean"}
      return {a.id:a.value for a in report.attributes if a.id in keys}
```

Changes fom the v10.2 document
Updated material on connecting to the SMRT Link Services API securely. (Pages 3-4)