PacBio

SMRT® Link Python API reference (v13.0)

Research use only. Not for use in diagnostic procedures.

P/N 103-285-100 Version 01 (December 2023)

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

SMRT® Link Python API reference (v13.0)

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 includes Python examples for using the API to perform the tasks listed in the Table of Contents.

Viewing API information

For detailed information on the SMRT Link web services API calls, including definitions and examples, see Swagger-generated information here:

`https://<server name>:8243/sl/docs/services/#/default`

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

Python reference client

SMRT Link v13.0 includes a new Python reference client covering the most common API methods. The Python reference client provides concise and self-documenting examples of how to use the PacBio API, and is easily portable to other languages. As the Python reference client is **not** a full SDK, it deals only with simple Python types (`int`, `str`, `float`, `list`, `dict`) rather than more complex objects.

The code for the Python reference client can be downloaded on github here.

The Python reference client is redistributable as a standalone library with no non-standard dependencies (only the `requests` library available on `PyPI`).

The Python reference client is **required** to use the Python examples in this document. The Python reference client includes methods for all of the following API calls:

Runs:
Get run(s)
Get run xml
Get run collection(s)
Get run from collection
Get run collection reports
Get run collection barcodes
Get run collection hifi reads
Get run collection hifi reads barcoded datasets
Get run reports
Get run design
Import run design CSV
Delete run
Import run xml
Update run xml

Chem bundle:
Get active bundle metadata
Get chemistry bundle metadata
Get active bundle file
Get chemistry bundle file

Jobs:
Get job
Get job reports
Download job report resources
Get job datastore
Get job entry points
Get job datasets
Get job options
Download job datastore file
Get analysis jobs
Get analysis jobs by state
Get analysis jobs by parent
Get SMRT Analysis nested jobs
Create analysis job
Terminate analysis job
Get import dataset jobs
Create import dataset [zip] job
Create import collection job
Create merge datasets job
Get pipeline(s)
Poll for successful job

Datasets:
Get consensus read sets
Get consensus read sets by movie
Get barcoded child datasets
Get subread sets
Get references
Get barcode sets
Get consensus read set
Get subread set
Get reference set
Get barcode set
Get consensus read set reports
Get barcode set contents
Get barcode set record names
Get dataset metadata
Get dataset jobs
Get dataset search

Python reference client setup instructions

If you downloaded the Python reference client to your local directory:

1. Make sure the `requests` module is installed for Python3**.** This will vary depending on your system, but `pip3 install requests` is one option.
2. Start the Python3 interpreter. (Version 3.9 or later is required.**)**
3. Import the Python reference client:
   `from smrtlink_client import SmrtLinkClient`
4. Set up the Python reference client with your SMRT Link server:
   `client = SmrtLinkClient.connect("<your-SMRT-Link-server-name>", "smrt-link-username", "password")`

If you have SMRT Link v13.0 installed, do the following:

1. Start the Python3 interpreter bundled with SMRT Link:
   `$SMRT_ROOT/smrtcmds/bin/python3`
2. Import the Python reference client:
   `from pbcommand.services.smrtlink_client import SmrtLinkClient`
3. Set up the Python reference client with your SMRT Link server:
   `client = SmrtLinkClient.connect("<your-SMRT-Link-server-name>", "smrt-link-username", "password")`

Security note: We recommend that you use a dedicated API client user that is **separate** from your usual network-wide login. SMRT Link administrators can define local users that only have access to the GUI and API **without** being able to log in to any other system.

How to query one or more runs and their details

You can find the run ID by using the `runs` endpoint and specifying a query to search for the desired run:

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

Following is an example of how to do this using Python:

```
runs = client.get_runs(name=None, reserved=None, instrumentType=None,
                       chipType=None, collectionUuid=None, movieName=None)
```

Example request:

```
runs = client.get_runs(instrumentType = "Revio")
```

In this example, you can query a run based on the following parameters:

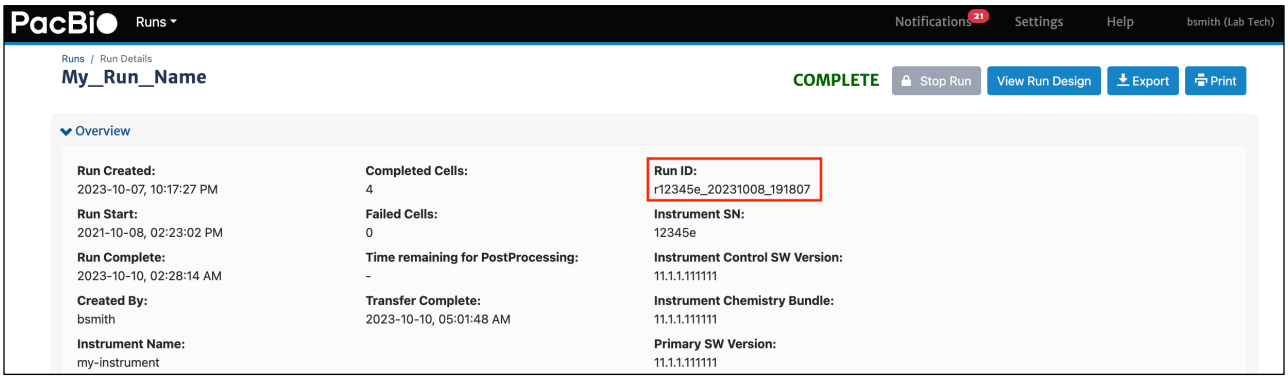
- `name (str)`: Filter by run name; partial matches are supported.
- `reserved (bool)`: Filter by reservation status; set to `true` for runs selected on instrument.
- `instrumentType (str)`: Filter by instrument type (`Revio`, `Sequel2e`, or `Sequel2`).
- `chipType (str)`: Filter by chip type (`8mChip` or `25mChip`).
- `collectionUuid (str)`: Retrieve the run for a specific collection UUID.
- `movieName (str)`: Filter by movie name associated with one of the runs.

Following is an example of what this returns:


```
{
"reserved": true, "numLRCells": 0,
"name": "20230414_Revio",
"completedAt": "2023-04-17T04:19:50.718Z",
"chemistrySwVersion": "1.2.3.11111",
"instrumentType": "Revio",
"chipType": "25mChip",
"instrumentName": "12345e",
"context": "r1234_20230414_212018",
"instrumentSwVersion": "1.1.1.11111",
"numCellsCompleted": 1,
"totalCells": 1,
"primaryAnalysisSwVersion": "12.0.0.0",
"status": "Complete",
"numStandardCells": 1,
"createdAt": "2023-04-14T18:17:23.849Z",
"startedAt": "2023-04-14T21:20:44.057Z",
"createdBy": "bobsmith",
"totalSamples": 97,
"numCellsFailed": 0,
"plate2": "12345678003015800739202310l8",
"instrumentSerialNumber": "12345",
"transfersCompletedAt": "2023-04-18T00:13:42.032Z",
"uniqueId": "0a12bcd2-5916-4142-9b82-220a7bb04d13",
"ccsExecutionMode": "OnInstrument"
}
```


- `reserved` means that the run design was used on an instrument. If `false`, that means the run design was created in SMRT Link, but has not yet been used in a run.
- `createdAt` is the time the run design was created.
- `startedAt` is the time the user pressed the button to close the Revio instrument door and begin a run.
- `completedAt` is the time when all of the instrument's movies have finished acquiring and **after** the cleanup; that is, instrument state = `Complete`, `Aborted`, or `Terminated`. The cleanup is skipped if another run has been pre-loaded.
- `uniqueID` is the Run UUID you see in the SMRT Link URL on the Run Details page:

- `context` is the Run ID you see in the SMRT Link GUI on the Run Details page:



How to retrieve all dataset IDs and movie names produced by a run

You can find all the datasets produced by a run by using the `runs` endpoint and specifying a query to search for collections with a matching run ID:

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

Following is an example of how to do this using Python:

```
collections = client.get_run_collections(run_id = "<run_id>")
```

Following is an example of what this returns:


```
[
  {
    "name": "Sequel2e_hifi_20231010_cell4",
    "completedAt": "2023-10-10T12:01:48.810Z",
    "instrumentName": "12345e",
    "context": "m64263e_211009_220021",
    "well": "D01",
    "projectId": 1,
    "sequencingKit": "123456101826100062822",
    "status": "Complete",
    "instrumentId": "12345e",
    "startedAt": "2021-10-09T22:00:22.030Z",
    "createdBy": "bobsmith",
    "cellType": "Standard",
    "uniqueId": "702d5232-9c2d-4f92-a35a-784949f9db0e",
    "collectionPathUri": "/data/hifiviral/r64263e_20211008_191807/4_D01",
    "ccsExecutionMode": "OnInstrument",
    "runId": "186f258a-dac7-479d-a122-aa0e6bbcab57",
    "ccsId": "9dbcaaf6-298a-465a-ad56-3175d1151c57",
    "movieMinutes": 480.0
  },
  {
    "name": "Sequel2e_hifiviral_20211010_cell4",
    "completedAt": "2021-10-10T04:39:54.955Z",
    "instrumentName": "12345e",
    "context": "m64263e_211009_134807",
    "well": "C01",
    "projectId": 1,
    "sequencingKit": "122894101826100062822",
    "status": "Complete",
    "importedAt": "2023-08-25T08:18:42.900Z",
    "instrumentId": "64263e",
    "startedAt": "2021-10-09T13:48:08.677Z",
    "createdBy": "bobsmith",
    "cellType": "Standard",
    "uniqueId": "c2690435-3336-4cb2-9bc1-b509d6901bbc",
    "collectionPathUri": "/data/hifiviral/r64263e_20211008_191807/3_C01",
    "ccsExecutionMode": "OnInstrument",
    "runId": "186f258a-dac7-479d-a122-aa0e6bbcab57",
    "ccsId": "ca24d2be-daee-480f-b3e6-c352104aff4d",
    "movieMinutes": 480.0
```

```
  },
...
]
```

- `ccsId` is the Dataset UUID that is visible at the top level of Data Management.
- `runId` is the Run ID you see in the URL in SMRT Link on the Runs/Run Details page:

- `context` is the movie name that is part of the output dataset you see on the Run Details page in SMRT Link.

How to query all the dataset details for a top-level dataset and all of its child datasets

Following is an example of how to do this using Python:

```
parent_dataset = client.get_consensusreadset(datset_id = <dataset_id>)
child_datasets = client.get_barcoded_child_datasets(parent_dataset_id = <ccsID>,
                                                    barcode_name=None,
                                                    biosample_name=None)
```

You can use this, for example, to find all the UUID's of your child datasets.

For Revio or Sequel IIe runs demultiplexed on-instruments, the parent dataset ID is the same as the `ccsId` referenced in the `collections` endpoint.

Following is an example of what this returns:

```
{
  "instrumentControlVersion": "1.1.1.111111",
  "tags": "ccs,testdata,barcoded",
  "instrumentName": "my-instrument",
  "uuid": "d717f468-e8f2-44cc-9a25-a50d699319b0",
  "dnaBarcodeName": "bc2082--bc2082",
  "totalLength": 1003747,
  "projectId": 1,
  "numRecords": 160,
  "wellSampleName": "My-sample_A01",
  "bioSampleName": "EC_40_82",
  "version": "3.0.1",
  "cellId": "DA120589",
  "id": 286,
  "md5": "c93c21ab91289d707e7adf31df57a949",
  "parentName": "My-sample_A01-Cell1 (CCS) (all samples)",
  "importedAt": "2023-08-25T12:16:50.898Z",
  "jobId": 164,
  "createdAt": "2023-08-25T12:13:48.489Z",
  "isActive": true,
  "createdBy": "bsmith",
  "wellName": "A01",
  "cellIndex": 0,
  "parentUuid": "f6dba83f-36b5-4443-b6fd-c8730ae350ad",
  "metadataContextId": "m64004_210910_192345",
```

```
  "numChildren": 0,
  "numResources": 1,
  "runName": "20231110_my_run",
  "datasetType": "PacBio.DataSet.ConsensusReadSet",
  "comments": "ccs dataset converted"
},
{
  "name": "My-sample_A01-Cell1 (CCS)(BS_43_85)",
  "updatedAt": "2023-08-25T12:13:48.560Z",
  "path": "<path>",
  "instrumentControlVersion": "1.1.1.111111",
  "tags": "ccs,testdata,barcoded",
  "instrumentName": "my-instrument",
  "uuid": "65ba42ac-a5f1-4f43-be2a-a26d32118dc4",
  "dnaBarcodeName": "bc2085--bc2085",
  "totalLength": 1176220,
  "projectId": 1,
  "numRecords": 146,
  "wellSampleName": "My-sample_A01",
  "bioSampleName": "BS_43_85",
  "version": "3.0.1",
  "cellId": "DA120589",
  "id": 285,
  "md5": "90607244e2d16c7035d6b652aa6be68a",
  "parentName": "My-sample_A01-Cell1 (CCS)(all samples)",
  "importedAt": "2023-08-25T12:16:50.851Z",
  "jobId": 164,
  "createdAt": "2023-08-25T12:13:48.560Z",
  "isActive": true,
  "createdBy": "bsmith",
  "wellName": "A01",
  "cellIndex": 0,
  "parentUuid": "f6dba83f-36b5-4443-b6fd-c8730ae350ad",
  "metadataContextId": "m64004_210910_192345",
  "numChildren": 0,
  "numResources": 1,
  "runName": "20231110_my_run",
  "datasetType": "PacBio.DataSet.ConsensusReadSet",
  "comments": "ccs dataset converted"
},
{
  "name": "My-sample_A01-Cell1 (CCS)(BS_40_81)",
  "updatedAt": "2023-08-25T12:13:48.456Z",
  "path": " <path>.consensusreadset.xml",
  "instrumentControlVersion": "1.1.1.111111",
  "tags": "ccs,testdata,barcoded",
  "instrumentName": "my-instrument",
  "uuid": "2b94dbb5-0be6-42b9-aa6c-a630c48a0e2d",
  "dnaBarcodeName": "bc2081--bc2081",
  "totalLength": 780298,
  "projectId": 1,
  "numRecords": 87,
  "wellSampleName": "My-sample_A01",
  "bioSampleName": "BS_40_81",
  "version": "3.0.1",
```

```
  "cellId": "DA120589",
  "id": 284,
  "md5": "2e34c910f9e9856d8215382024e34109",
  "parentName": "My-sample_A01-Cell1 (CCS) (all samples)",
  "importedAt": "2023-08-25T12:16:50.804Z",
  "jobId": 164,
  "createdAt": "2023-08-25T12:13:48.456Z",
  "isActive": true,
  "createdBy": "bsmith",
  "wellName": "A01",
  "cellIndex": 0,
  "parentUuid": "f6dba83f-36b5-4443-b6fd-c8730ae350ad",
  "metadataContextId": "m64004_210910_192345",
  "numChildren": 0,
  "numResources": 1,
  "runName": "20231110_my_run ",
  "datasetType": "PacBio.DataSet.ConsensusReadSet",
  "comments": "ccs dataset converted"
},
...
]
```

How to query P0, P1, and P2 metrics

You can query P0, P1, and P2 loading metrics using the SMRT Link API. The following example searches run collection reports for the Loading Report and extracts the P0, P1, and P2 values from it. You need the Run UUID and Collection UUID.

Following is an example of how to do this using Python:

```
reports = client.get_run_collection_reports(<run_uuid>, <collection_uuid>)
for r in reports:
       if "loading" in r["reportTypeId"]:
              report_uuid = r["dataStoreFile"]["uuid"]
              report = client.load_datastore_report_file(report_uuid)
              print({attr["id"]:attr["value"] for attr in report["attributes"]})
```

Following is an example of what this returns:

```
{'loading_xml_report.productive_zmws': 8014671,
'loading_xml_report.productivity_0_n': 808345,
'loading_xml_report.productivity_1_n': 6161738,
'loading_xml_report.productivity_2_n': 1044588}
```

How to export demultiplexed datasets

You can retrieve the demultiplexed "child" datasets for a PacBio instrument run, optionally filtering by barcode name (such as `bc2001--bc2001`) or biosample name.

Note: Biosample name = well name = well sample name = collection name.

Following is an example of how to do this using Python:

```
#first get the collection_id
collection_id = client.get_run_collections(<run_id>)
```

This returns:


```
  {
    "name": "20230414_84026_16kbHG002_97barcodes ",
    "completedAt": "2023-04-18T00:13:42.032Z",
    "instrumentName": "my-instrument",
    "context": "m84026_230415_224020_s3",
    "well": "C01",
    "projectId": 1,
    "sequencingKit": "03015810211880010182 3",
    "labelNumber": "00739",
    "status": "Complete",
    "importedAt": "2023-04-17T23:55:12.727Z",
    "instrumentId": "84026",
    "startedAt": "2023-04-15T22:40:24.006Z",
    "createdBy": "admin",
    "cellType": "Standard",
    "uniqueId": "69b09865-5c23-4717-92ad-75968a43f443",
    "collectionPathUri": "/collections/349/r84026_20230414_212018/2_C01",
    "ccsExecutionMode": "OnInstrument",
    "runId": "0f99fea6-5916-4142-9b82-220a7bb04d13",
    "ccsId": "3acce2c3-d904-4d08-aba0-2628d0dcccbf",
    "movieMinutes": 1440.0
  }
  '''

 #uniqueId is the collection_id

demuxed_datasets = client.get_run_collection_hifi_reads_barcoded_datasets(<run_id>,
<collection_id>,
barcode_name=None,
biosample_name=None)
```


Following is an example of what this returns:


```
[
  {
    "name": "20230414_84026_225pM_97Barcodes_bsmith",
    "updatedAt": "2023-04-17T23:55:12.721Z",
    "path": "<path>.consensusreadset.xml",
    "instrumentControlVersion": "1.1.1.11111",
    "tags": "ccs,barcoded",
    "instrumentName": "my-instrument",
    "uuid": "ab1234cd-5efg-7h99-0000-0a1209170a1b",
    "dnaBarcodeName": "bc2096--bc2096",
    "totalLength": 891005473,
    "projectId": 1,
    "numRecords": 56787,
    "wellSampleName": "my_wellsamplename",
    "bioSampleName": "my_biosamplename",
    "version": "3.0.1",
    "cellId": "10000004751022022000714 23",
    "id": 132,
    "md5": "33c8697d37a3feb9fbb844c913ea916c",
    "parentName": "20230414_84026_225pM_97Barcodes",
    "importedAt": "2023-04-17T23:55:12.721Z",
    "jobId": 27,
```

```
    "createdAt": "2023-04-17T23:55:12.721Z",
    "isActive": true,
    "createdBy": "bsmith",
    "wellName": "C01",
    "cellIndex": 6,
    "parentUuid": "3acce2c3-d904-4d08-aba0-2628d0dcccbf",
    "metadataContextId": "m12345_230415_224020_s3",
    "numChildren": 0,
    "numResources": 1,
    "runName": "20230414_84026_16kbHG002_97Barcodes",
    "datasetType": "PacBio.DataSet.ConsensusReadSet",
    "comments": "Record generated by runqc-reports"
  },
  {
    "name": "20230414_84026_16kbHG002_97Barcodes-Cell7 (225pM_16kb_HG002)",
    "updatedAt": "2023-04-17T23:55:12.647Z",
    "path": "<path>.consensusreadset.xml",
    "instrumentControlVersion": "1.1.1.11111",
    "tags": "ccs,barcoded",
    "instrumentName": "my-instrument",
    "uuid": "7960507a-437a-455b-a1b0-9cc6a9c2386d",
    "dnaBarcodeName": "bc2095--bc2095",
    "totalLength": 553068822,
    "projectId": 1,
    "numRecords": 37200,
    "wellSampleName": "20230414_84026_16kbHG002_97Barcodes",
    "bioSampleName": "225pM_16kb_HG002",
    "version": "3.0.1",
    "cellId": "10000004751022022000071423",
    "id": 131,
    "md5": "f21fcf132a20e009b13836cfc403ce15",
    "parentName": "20230414_84026_16kbHG002_97Barcodes-Cell7 (all samples)",
    "importedAt": "2023-04-17T23:55:12.647Z",
    "jobId": 27,
    "createdAt": "2023-04-17T23:55:12.647Z",
    "isActive": true,
    "createdBy": "admin",
    "wellName": "C01",
    "cellIndex": 6,
    "parentUuid": "3acce2c3-d904-4d08-aba0-2628d0dcccbf",
    "metadataContextId": "m84026_230415_224020_s3",
    "numChildren": 0,
    "numResources": 1,
    "runName": "20230414_84026_16kbHG002_97Barcodes",
    "datasetType": "PacBio.DataSet.ConsensusReadSet",
    "comments": "Record generated by runqc-reports"
  },
...
]
```

How to query datasets

You can retrieve a list of HiFi datasets, with optional search parameters.

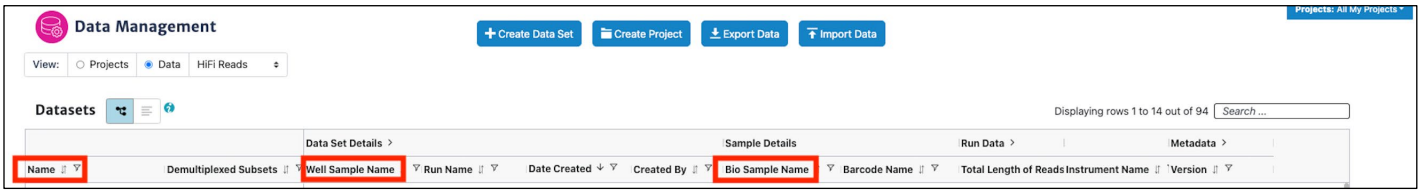
Following is an example of how to do this using Python:

```
client.get_consensusreadsets(name=None,
                             bioSampleName=None,
                             wellSampleName=None,
                             metadataContectId=None)
```

Following is a partial list of supported search terms:

- `name` (dataset name)
- `bioSampleName`
- `wellSampleName`
- `metadataContextId` (movie name)

You can find the `name` (dataset name), `wellSampleName`, and `bioSampleName` in the SMRT Link GUI here:



You can find the Movie Name by clicking the dataset name and viewing the Dataset Details page.

Search term notes:

- String searches are always **case-insensitive**.
- Most of the non-timestamp string fields in the data model are searchable using partial strings by adding the prefix `like:` to the search term, such as `client.get_consensusreadsets(bioSampleName="like:HG002")`
- The prefixes `not:` (inequality), `unlike:`, `start:` and `end:` are also supported.
- For numerical fields, `not:`, `lt:`, `lte:`, `gt:`, and `gte:` are supported, as well as `range:{start},{end}`.

How to terminate a SMRT Link job

You can immediately terminate an analysis job.

Following is an example of how to do this using Python:

```
client.terminate_analysis_job(job_id)
```

This returns:

```
{'message': 'Cromwell workflow <uuid> TERMINATED for <job_id>'}
```

How to get all created, running, successful, failed, or terminated analysis jobs

You can search for all analysis jobs of a specific state: `CREATED`, `SUBMITTED`, `RUNNING`, `SUCCESSFUL`, `FAILED`, `TERMINATED`, or `ABORTED`.

Following is an example of how to do this using Python:

```
terminated_jobs = client.get_analysis_jobs_by_state(state)
```

Following is an example that returns all jobs that were terminated:

```
terminated_jobs = client.get_analysis_jobs_by_state(state = 'TERMINATED')
```

Following is an example of what this returns:


```
[{
  "subJobTypeId": "cromwell.workflows.pb_assembly_hifi",
  "name": "test",
  "updatedAt": "2023-09-20T07:51:05.637Z",
  "workflow": "{}",
  "path": "<path>",
  "state": "TERMINATED",
  "tags": "",
  "uuid": "1b717f91-6651-45d3-b8cf-e1b4b4427f6a",
  "externalJobId": "547075f8-674d-4fbb-a0ba-69f55c627892",
  "jobStartedAt": "2023-09-20T07:50:10.373Z",
  "applicationName": "Genome Assembly",
  "projectId": 1,
  "childJobsCount": 0,
  "jobCompletedAt": "2023-09-20T07:51:05.637Z",
  "jobTypeId": "analysis",
  "id": 627,
  "smrtlinkVersion": "1.1.1.11111",
  "comment": "Description for job Run Analysis Application",
  "isNested": false,
  "createdAt": "2023-09-20T07:50:05.567Z",
  "isActive": true,
  "createdBy": "bsmith",
  "createdByEmail": "bsmith@company.com",
  "isMultiJob": false,
  "jsonSettings": "",
  "jobUpdatedAt": "2023-09-20T07:51:05.637Z"
}]
```

How to poll for a job to successfully complete within a specified timeout

You can poll a submitted job of any type until it completes successfully within the specified timeout, or raise an exception.

Following is an example of how to do this using Python:

```
client.poll_for_successful_job(job_id, sleep_time, max_time):
```

Following is an example of what this returns:

```
finished_job = client.poll_for_successful_job(job_id=<job_id>,
                                              sleep_time=120,
                                              max_time=28800)
```

How to query a Run XML

You can retrieve the XML data model for a PacBio instrument run.

Following is an example of how to do this using Python:

```
run_xml = client.get_run_xml(run_id)
```

This returns the XML as a string; this is the primary format that SMRT Link uses to send instructions to the instrument, and much of it will end up in the output dataset XMLs (with further modifications from the instrument software).

How to query a run design

Following is an example of how to do this using Python:

```
run_design = client.get_run_design(run_id)
```

Following is an example of what this returns for an AAV run:


```
{
  "chipType": "25mChip",
  "createdBy": "bsmith",
  "experimentDescription": "",
  "experimentId": "",
  "experimentName": "",
  "instrumentType": "Revio",
  "runDescription": "",
  "runName": "Bsmith Run 09.28.2023 18:27",
  "samples": [
    {
      "application": "AAVAnalysis",
      "includeCpG": true,
      "isBarcoded": true,
      "sampleDescription": "",
      "readSegmentation": false,
      "demuxMode": "OnInstrument",
      "uuid": "9aaee069-8ecf-4b4f-8f37-d0bd6a3abb5a",
      "ccsUuid": "0fab0fc2-4c13-496c-aa06-352a30539e6f",
      "includeLowQuality": true,
```

```
      "barcodeCsvFileName": "",
      "sampleName": "test-well",
      "ccsMode": "OnInstrument",
      "includeKinetics": false,
      "useDynamicLoading": true,
      "loadingConcentration": 0.0,
      "projectId": 1,
      "adapter": "mas16",
      "barcodedSamples": [
        {
          "bioSampleName": "Bio Sample 1",
          "dnaBarcodeName": "bc2001--bc2001",
          "uuid": "560148e0-c4a4-4512-9e2c-2c3a89a80f6d"
        }
      ],
      "scIsoSeq": false,
      "controlKit": "Lxxxxx102798000123199",
      "templatePrepKit": "Lxxxxx999999001123199",
      "movieTimeHours": 24.0,
      "barcodesFasta":
">bc2001\nATCGTGCGACGAGTAT\n>bc2002\nTGCATGTCATGAGTAT\n>bc2003\nACGAGTGCTCGAGTAT\n>bc200
4\nTGCAGTGCTCGAGTAT\n>bc2005\nTGACTCGATCGAGTAT\n>bc2006\nCATGCGATCTGAGTAT\n>...\n",
      "libraryType": "AAV",
      "automationParameters": [
        {
          "name": "MovieLength",
          "valueDataType": "Double",
          "simpleValue": "1440"
        }
      ],
      "sequencingKit": "000000102118800110723",
      "labelNumber": "12345",
      "primaryAutomationName": "",
      "emitSubreadsPercent": 0,
      "copyFiles": [],
      "barcodeUuid": "43f950a9-8bde-3855-6b25-c13368069745",
      "insertSize": 500,
      "wellName": "A01",
      "consensusMode": "strand",
      "symmetricBarcodes": true,
      "cellType": "human",
      "minBarcodeScore": 80,
      "heteroduplexDetection": false,
      "plateNumber": 1,
      "bindingKit": "Lxxxxx102739100123123"
    }
  ],
  "uuid": "ed271e1b-efa4-4999-ab7f-a945350348c3"
}
```


How to create a run design by importing a run design CSV

Following is an example of how to do this using Python:

```
client.import_run_design_csv(csv_file)
```

How to find the Run QC reports associated with an analysis job

You can obtain **all** collection-level reports associated with a run. (**Note**: This was introduced in SMRT Link v13.0.)

Following is an example of how to do this using Python:

```
run_reports = client.get_run_reports(run_id)
```

This returns information about the following reports, including when the report was created and the path to the report. (**Note**: To get the **content** of the reports, see **How to find the contents of specific Run QC reports**.)

- import_dataset.report_detect_cpg_methyl
- collection.barcode_preview_report
- import_dataset.report_adapters
- import_dataset.report_loading
- import_dataset.report_raw_data
- import_dataset.report_barcode
- import_dataset.report_control
- import_dataset.report_ccs2
- import_dataset.report_ccs_processing

Following is an example of what this returns:


```
[
  {
    "dataStoreFile": {
      "createdAt": "2023-09-28T08:21:56.426Z",
      "description": "detect_cpg_methyl",
      "fileSize": 1420,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-09-28T08:22:03.997Z",
      "isActive": true,
      "modifiedAt": "2023-09-28T08:21:56.426Z",
      "name": "Report detect_cpg_methyl",
      "path": "<directories>/reports/detect_cpg_methyl.report.json",
      "sourceId": "import_dataset.report_detect_cpg_methyl",
      "uuid": "4229c152-4e49-4199-adaa-d6d91ffd83b3"
    },
    "reportTypeId": "import_dataset.report_detect_cpg_methyl"
  },
  {
    "dataStoreFile": {
      "createdAt": "2023-09-28T08:16:51.670Z",
      "description": "PacBio Report barcode_preview_report (6ad9e5df-5aaf-44d3-9499-
5ad7a0cc163d)",
      "fileSize": 5381,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-09-28T08:16:51.683Z",
      "isActive": true,
      "modifiedAt": "2023-09-28T08:16:51.670Z",
      "name": "Barcode Preview Report",
      "path": "<path>/barcode_preview_report.report.json",
      "sourceId": "collection.barcode_preview_report",
      "uuid": "6ad9e5df-5aaf-44d3-9499-5ad7a0cc163d"
```

```
    },
    "reportTypeId": "collection.barcode_preview_report"
  },
  {
    "dataStoreFile": {
      "createdAt": "2023-09-28T08:21:56.448Z",
      "description": "PacBio Report adapter_xml_report (d340b519-6fdc-401a-b3b0-
731245ec6c26)",
      "fileSize": 836,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-09-28T08:22:03.998Z",
      "isActive": true,
      "modifiedAt": "2023-09-28T08:21:56.448Z",
      "name": "Adapter Report",
      "path": "<path> /adapter.report.json",
      "sourceId": "import_dataset.report_adapters",
      "uuid": "d340b519-6fdc-401a-b3b0-731245ec6c26"
    },
    "reportTypeId": "import_dataset.report_adapters"
  },
  {
    "dataStoreFile": {
      "createdAt": "2023-09-28T08:21:56.451Z",
      "description": "PacBio Report loading_xml_report (a33dfa8c-e591-453d-a9e2-
d4f6aab379b6)",
      "fileSize": 4106,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-09-28T08:22:03.998Z",
      "isActive": true,
      "modifiedAt": "2023-09-28 <path> /loading.report.json",
      "sourceId": "import_dataset.report_loading",
      "uuid": "a33dfa8c-e591-453d-a9e2-d4f6aab379b6"
    },
    "reportTypeId": "import_dataset.report_loading"
  },
  {
    "dataStoreFile": {
      "createdAt": "2023-09-28T08:21:56.462Z",
      "description": "PacBio Report raw_data_report (924e472c-caa0-4d3d-bf10-
df09b850915b)",
      "fileSize": 3390,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-09-28T08:22:03.999Z",
      "isActive": true,
      "modifiedAt": "2023-09-28T08:21:56.462Z",
      "name": "Raw Data Report",
      "path": "<path>/raw_data.report.json",
      "sourceId": "import_dataset.report_raw_data",
      "uuid": "924e472c-caa0-4d3d-bf10-df09b850915b"
    },
    "reportTypeId": "import_dataset.report_raw_data"
  },
  {
    "dataStoreFile": {
      "createdAt": "2023-09-28T08:21:56.619Z",
      "description": "PacBio Report barcode (63facc29-49f2-4df8-84df-75711813efac)",
```

```
      "fileSize": 33184,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-09-28T08:22:04.000Z",
      "isActive": true,
      "modifiedAt": "2023-09-28T08:21:56.619Z",
      "name": "Report barcode",
      "path": "<path> /barcodes.report.json",
      "sourceId": "import_dataset.report_barcode",
      "uuid": "63facc29-49f2-4df8-84df-75711813efac"
    },
    "reportTypeId": "import_dataset.report_barcode"
  },
  {
    "dataStoreFile": {
      "createdAt": "2023-09-28T08:21:56.600Z",
      "description": "PacBio Report control (2155f2a7-dfa9-4cc2-8e14-b7677406a8ae)",
      "fileSize": 2155,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-09-28T08:22:03.999Z",
      "isActive": true,
      "modifiedAt": "2023-09-28T08:21:56.600Z",
      "name": "Control Report",
      "path": "<path>/control.report.json",
      "sourceId": "import_dataset.report_control",
      "uuid": "2155f2a7-dfa9-4cc2-8e14-b7677406a8ae"
    },
    "reportTypeId": "import_dataset.report_control"
  },
    {
    "dataStoreFile": {
      "createdAt": "2023-10-25T08:23:00.230Z",
      "description": "PacBio Report ccs2 (e4363794-b72b-40bf-b881-180bf733416a)",
      "fileSize": 9777,
      "fileTypeId": "PacBio.FileTypes.JsonReport",
      "importedAt": "2023-10-25T08:23:16.895Z",
      "isActive": true,
      "modifiedAt": "2023-10-25T08:23:00.230Z",
      "name": "CCS Analysis Report",
      "path": "<path>/reports/ccs.report.json",
      "sourceId": "import_dataset.report_ccs2",
      "uuid": "e4363794-b72b-40bf-b881-180bf733416a"
    },
    "reportTypeId": "import_dataset.report_ccs2"
  }
]
```

How to find the contents of specific Run QC reports

You can obtain the contents of the following Run QC reports:

- **Detect CpG Methylation Report**
- **Barcode Preview Report**
- **Adapter Report**
- **Loading Report**
- **Raw Data Report**
- **Report barcode**
- **Control Report**
- **CCS Analysis Report**

Following is an example of how to do this using Python:

```
reports = client.get_run_reports(run_id)
```

For `r` in reports:

```
      report_uuid = r["dataStoreFile"]["uuid"]
      report = client.load_datastore_report_file(report_uuid)
```

How to query instrument status, such as Running or Complete

You can obtain the instrument state of an instrument using the instrument's serial number.

The possible instrument states are:
`Starting`, `WarmUp`, `SelfTest`, `Ready`, `Running`, `ShuttingDown`, `Problem`. (`Ready` indicates that the prior run was complete and the instrument is ready to begin sequencing again.)

Following is an example of how to do this using Python:

```
instrument_state = client.get_instrument_state(serial)
```

To obtain the instrument states of **all** your instruments:

```
instrument_states = client.get_instrument_states()
```

How to query a job

You can retrieve a job of any type by integer ID or UUID.

Following is an example of how to do this using Python:

```
job = client.get_job(job_id)
```

Following is an example of what this returns:

```
{
  "subJobTypeId": "cromwell.workflows.pb_segment_reads",
  "name": "Read Segmentation with multiple cells",
  "updatedAt": "2023-09-28T08:28:49.099Z",
  "workflow": "{}",
  "path": "<path>",
  "state": "SUCCESSFUL",
```

```
  "tags": "testkit",
  "uuid": "d6e7b447-4af2-4fb2-9b8b-4188a930aa28",
  "externalJobId": "d698e9f9-c521-449d-961d-b7946931ea5f",
  "jobStartedAt": "2023-09-28T08:24:36.986Z",
  "applicationName": "Read Segmentation",
  "projectId": 1,
  "childJobsCount": 0,
  "jobCompletedAt": "2023-09-28T08:28:49.099Z",
  "jobTypeId": "analysis",
  "id": 131,
  "smrtlinkVersion": "1.1.1.11111",
  "comment": "Description for job Run Analysis Application",
  "isNested": false,
  "createdAt": "2023-09-28T08:24:35.732Z",
  "isActive": true,
  "createdBy": "bsmith",
  "createdByEmail": "bsmith@comp[any.com",
  "isMultiJob": false,
  "jsonSettings": "{\"name\":\"Read Segmentation with multiple cells\"...}",
  "jobUpdatedAt": "2023-09-28T08:28:49.099Z"
}
```


How to start a SMRT Link job for a specific workflow

Following is an example of how to do this using Python:

```
job = client.create_analysis_job(options)
```

You can submit a SMRT Analysis job to be run as soon as possible. (This requires that **all** input datasets have already been imported.)

Options:

Job options schema:

- `pipelineId`: String such as `'cromwell.workflows.pb_align_ccs'`.
- `name`: Job name string.
- `entryPoints`: List of dataset entry points.
- `taskOptions`: List of workflow task options.
- `projectId`: `int` or `null`.
- `presetId`: `string` or `null`.

Entry point model:

- `entryId`: Pre-set identifier, can be any of `eid_ccs`, `eid_barcode`, `eid_ref_dataset`, `eid_barcode_2`, or `eid_subread`.
- `fileTypeId`: Dataset MetaType, from the top-level XML tag.
- `datasetId`: Dataset UniqueID (UUID).

Task/workflow option model:

- `optionId`: String ID such as `mapping_min_length`.
- `value`: string, float, int, bool, or occasionally null.
- `optionTypeId`: Type of `value` field.

How to combine a sample split across multiple cells

Following is an example of how to do this using Python:

```
DS_TYPE = "PacBio.DataSet.ConsensusReadSet"
datasets = client.get_consensusreadsets(bioSampleName="MySample1234")
job = client.create_merge_datasets_job([d["uuid"] for d in datasets])
job = client.poll_for_successful_job(job["id"])
datastore = client.get_job_datastore(job["id"])
merged_datasets = [f for f in datastore if f["fileTypeId"] == DS_TYPE]
merged_datasets = client.get_consensusreadsets(jobId=job["id"])
```

How to poll every 10 minutes until a collection is complete, then launch a HiFi Mapping job using the official PacBio hg38 reference, and poll until it completes successfully

Following is an example of how to do this using Python:

```
import time
collection = client.get_run_collection(run_id, collection_id)
while True:
    dataset = client.get_dataset_search(collection["ccsId"])
    if dataset:
        break
    else:
        time.sleep(600)
job = client.create_analysis_job({
    "name": "My Mapping Job",
    "pipelineId": "cromwell.workflows.pb_align_ccs",
    "entryPoints": [
        {
            "entryId": "my_id",
            "datasetId": collection["ccsId"],
            "fileTypeId": "PacBio.DataSet.ConsensusReadSet"
        },
        {
            "entryId": "my_id_ref_dataset",
            "datasetId": "ab1234cd-5efg-7h99-0000-0a1209170a1b",
            "fileTypeId": "PacBio.DataSet.ReferenceSet"
        }
    ],
    "taskOptions": []
})
job = client.poll_for_successful_job(job["id"])
```

How to find information about the Run QC reports associated with an analysis job

You can obtain **all** collection-level reports associated with a run. (**Note**: This was Introduced in SMRT Link 13.0.)

Following is an example of how to do this using Python:

```
entry_points = client.get_job_entry_points(job_id)
movie_names = set([])
for entry_point in entry_points:
    if entry_point["datasetType"] == "PacBio.DataSet.ConsensusReadSet":
        dataset = client.get_consensusreadset(entry_point["datasetUUID"])
        movie_names.append(dataset["metadataContextId"])
qc_reports = []
for movie_name in movie_names:
    runs = client.get_runs(movieName=movie_name)
    if len(runs) == 1:
        collections = client.get_run_collection(runs[0]["unique_id"])
        for collection in collections:
            if collection["context"] == movie_name:
                reports = client.get_collection_reports(run_id,
                    collection["unique_id"])
                qc_reports.append(reports)
```

How to export demultiplexed dataset metrics as a CSV file

Following is an example of how to do this using Python. **Note**: This **also** requires the `pbcommand` library distributed with SMRT Link, but it is also possible to extract these data data from the raw report JSON.

```
from pbcommand.pb_io import load_report_from_json
from pbcommand.services.smrtlink_client import SmrtLinkClient

def export_barcodes_report_csv(server, user, password, dataset_uuid, output_file):
    client = SmrtLinkClient.connect (server, user, password)
    reports = client.get_consensusreadset_reports(dataset_uuid)
    for report_file in reports:
        if "barcode" in report_file["reportTypeId"]:
            rpt_uuid = report_file["dataStoreFile"]["uuid"]
            report_json = client.load_datastore_report_file(rpt_uuid)
            report = load_report_from_json(report_json)
            report.tables[0].to_csv(output_file)
            break
    else:
        raise RuntimeError(f"Can't find barcodes report for {dataset_uuid}")
```