Analysis Services API Tutorial

Introduction

This tutorial is about the Analysis Services API (Application Programming Interface). It assumes some knowledge of REST APIs, HTTP, and having necessary skills to develop software applications that consume APIs.

This tutorial is primarily focused on the aspects of the API required to execute models via "workflows", for discussion of how to implement and install customised models, please refer to the Model Developer's Guide.

The API

The Analysis Services API is a RESTful service adopting the HAL specification (http://stateless.co/hal_specification.html). The API endpoint can be found here: https://senaps.io/api/analysis (https://senaps.io/api/analysis).

For an interactive API interface you can explore the API docs (https://senaps.io/api-docs/).

The API lets you create, query and manage resources via:

`/`

The root resource as the entry point into the API.

`/base-images`

A base image provides a foundation upon which customised models can be built. These are only of interest to people developing custom models, further discussion can be found in the Model Developer's Guide.

`/models`

A model is a configurable computational process that processes input data to produce output data.

`/workflows`

A workflow defines the input, output and configuration parameters for running a model.

`/jobs`

A job provides previous and current workflow execution details

`/schedules`

Schedules provide timed execution of workflows.

`/documentnodes`

A document node contains configuration values for a workflow.

Registering

Registering new accounts can be done via the Senaps data portal web site. To do this go to: https://senaps.io/dashboard/#/access/signup (https://senaps.io/dashboard/#/access/signup).

Initially new users don't have any specific authorisation, they will only be able to see data sets which have been shared publicly (none, as of April 2016).

There are two ways of gaining a role in the platform:

1. Contact site admin to ask for a new organisation to be created.
2. Ask an existing user with sufficient permissions to give you a role in their organisation. Organisation managers have enough permissions to assign roles to new users.

Accessing the API

All API requests must be authenticated using HTTP Basic Authentication (https://en.wikipedia.org/wiki/Basic_access_authentication). Most HTTP client libraries provide support for basic authentication.

The API is secured over HTTPS using SHA-256 with RSA Encryption.

Postman (https://www.getpostman.com/) is a useful program that lets you construct API requests and generate code examples for consuming the APIs in popular programming languages.

Required HTTP Headers

All API requests must include the following headers.

```
Content-Type: application/json
Authorization: Basic <<username:password base-64 encoded>>
```

Base-64 Encoding Username and Password

For basic authentication to work you need provide a base-64 encoded username and password, separated by a colon ':'. Most programming languages provide support for base-64 encoding. There *are* online tools that can convert plain text to base-64, however submitting sensitive details to non-secure or untrusted websites is not recommended.

As a concrete example: given a user with username `user@example.com` and with password `test`, first we concatenate the username, a colon, and the password together to form `user@example.com.au:test`, which is then base-64 encoded to result in `dXNlckBleGFtcGxlLmNvbS5hdTp0ZXN0`.

This value is then added to the HTTP `Authorization` header:

```
Content-Type: application/json
Authorization: Basic dXNlckBleGFtcGxlLmNvbS5hdTp0ZXN0
```

Models

Conceptually, a model is an implementation of an algorithm that performs some useful computation on a set of inputs, and produces a set of outputs. Models often also accept some number of configuration parameters to allow their behaviour to be customised.

Within the Analysis Services API, those inputs, outputs and configuration parameters are described by "ports", which form the interface between the model and its users.

Installing Models

Custom models can be installed by making a `POST` request to the `/models` endpoint. This is described in full detail in a dedicated section of the Model Developer's Guide.

Retrieving Models

A list of the currently available models can be retrieved by issuing a `GET` request to the API's `/models` endpoint.

This endpoint supports paginated requests using the `skip` and `limit` query parameters. Use `skip` to skip past a number of models in the overall list, and use `limit` to limit the number of models returned by the request. For example, if paginating with 20 models per page, the third page can be retrieved by setting `skip=40` and `limit=20`. If omitted, `skip` defaults to zero, and `limit` defaults to 1000.

For example, to retrieve a page containing three models:

```
GET /api/analysis/models?skip=3&limit=3 HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base-64 encoded>>

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/hal+json

{
  "skip": 3,
  "limit": 3,
  "count": 3,
  "totalcount": 18,
  "_links": {
  "next": {
    "href": "https://senaps.io/api/analysis/models/?skip=6&limit=3"
  },
  "previous": {
```

```
      "href": "https://senaps.io/api/analysis/models/?skip=0&limit=3"
    },
    "last": {
      "href": "https://senaps.io/api/analysis/models/?skip=15&limit=3"
    },
    "self": {
      "href": "https://senaps.io/api/analysis/models/?skip=3&limit=3"
    },
    "first": {
      "href": "https://senaps.io/api/analysis/models/?skip=0&limit=3"
    }
    },
    "_embedded": {
    "models": [
      {
      "id": "percentile",
      "name": "percentile",
      "version": "1.0",
      "organisationid": "csiro",
      "groupids": [],
      "description": "calculate 25,50 and 75 percentile for McCall model forecasts",
      "_links": {
        "self": {
        "href": "https://senaps.io/api/analysis/models/percentile"
        }
      }
      },
      {
      "id": "DairyMod",
      "name": "DairyMod",
      "version": "1.0",
      "organisationid": "csiro",
      "groupids": [],
      "_links": {
        "self": {
        "href": "https://senaps.io/api/analysis/models/DairyMod"
        }
      }
      },
      {
      "id": "multivariate_mean",
      "name": "Multivariate Mean",
      "version": "0.0.1",
      "organisationid": "csiro",
      "groupids": [],
      "description": "Computes mean of multiple aligned data streams.",
      "method": "Downloads observation data for inputs, scales each stream by the selected weighting, computes sum of all observatio
      "_links": {
        "self": {
        "href": "https://senaps.io/api/analysis/models/multivariate_mean"
        }
      }
      }
    ]
    }
}
```


The returned document consists of the following properties:

Property	Description
`skip`	The value of the `skip` query parameter.
`limit`	The value of the `limit` query parameter.
`count`	The number of models in the current "page" of results.
`totalcount`	The total number of models.
`_links.self.href`	The URL of the current "page" of results.

`_links.first.href`	The URL of the first "page" of results. Omitted if the current page is the first page.
`_links.last.href`	The URL of the last "page" of results. Omitted if the current page is the last page.
`_links.previous.href`	The URL of the previous "page" of results. Omitted if the current page is the first page.
`_links.next.href`	The URL of the next "page" of results. Omitted if the current page is the last page.
`_embedded.models`	A list of the models on the current "page" (see the next table following).

Each model in the `_embedded.models` list has the following properties:

Property	Description
`id`	The model's unique ID, used to refer to it elsewhere within the API.
`name`	The model's human-friendly name.
`version`	The model's version.
`organisationid`	The ID of the Senaps organisation that owns the model.
`groupids`	The ID(s) of the Senaps groups that own the model within the organisation.
`description`	A description of what the model does.
`method`	A description of how the model works.
`_links.self.href`	The model's unique API URL.

Note that some optional properties (in particular the `description` and `method`) may be omitted from the document if not specified for the model.

A model's unique URL consists of its ID appended to the `/models` endpoint URL. Performing a `GET` request on the model's URL will return a more detailed representation of the model.

For example, the following request retrieves the detailed representation of the `multivariate_mean` model:

```
GET /api/analysis/models/multivariate_mean HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base-64 encoded>>

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/hal+json

{
  "id": "multivariate_mean",
  "name": "Multivariate Mean",
  "version": "0.0.1",
  "description": "Computes mean of multiple aligned data streams.",
  "method": "Downloads observation data for inputs, scales each stream by the selected weighting, computes sum of all observations
  "organisationid": "csiro",
  "groupids": [],
  "_links": {
  "self": {
    "href": "https://senaps.io/api/analysis/models/multivariate_mean"
  }
  },
  "_embedded": {
  "ports": [
    {
    "portname": "output",
    "required": true,
    "type": "stream",
    "description": "The stream to place the averaged data into.",
    "direction": "output"
    },
    {
    "portname": "inputs",
    "required": true,
    "type": "multistream",
    "description": "The streams to be averaged",
    "direction": "input"
    },
    {
    "portname": "weights",
    "required": false,
    "type": "document",
    "description": "The weighting of each stream, given as a JSON object of stream_id -> weight pairs. If no weight specified for
    "direction": "input"
    }
  ]
  }
}
```

In addition to the properties included in the response to the `GET /models` request, directly requesting a specific model using the `GET /models/<model_id>` request also returns a list of the model's input and output ports in the `_embedded.ports` property. Each of those ports has the following properties:

Property	Description
`portname`	The name of the port.
`required`	Whether a value must be provided for the port when executing the model. Either `true` or `false`.
`type`	The port type, one of: • `stream` for ports that accept a single Senaps data stream • `multistream` for ports that accept multiple Senaps data streams • `document` for ports that accept an arbitrary text document • `grid` for ports that accept gridded data from Thredds

`description`	A human-friendly description of the port.
`direction`	Whether the port is an input or an output port.

Workflows

Workflows are the means by which specific inputs and outputs can be attached to a model in order to run it. They achieve this by binding specific Senaps stream IDs, grid datasets, and/or text documents to the model's ports.

Creating Workflows

Workflows are created by executing a `POST` request on the API's `/workflows` endpoint, with the request body containing a "workflow creation" document.

For example, the following request would create a new workflow for the `multivariate_mean` model:

```
POST /api/analysis/workflows HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base-64 encoded>>

{
  "name": "New Multivariate Mean workflow",
  "description": "Another Multivariate Mean workflow",
  "organisationid": "csiro",
  "groupids": [],
  "runas": { "roles": [ "my_example_role" ]}
  "graph": {
    "nodes": [
      {
        "id": "multivariate_mean",
        "label": "multivariate_mean",
        "operatorid": "multivariate_mean"
      },
      {
        "id": "output_node",
        "label": "output",
        "streamid": "epa.HT.Tave"
      },
      {
        "id": "inputs_node",
        "label": "inputs",
        "streamids": [
          "epa.HT.Tin",
          "epa.HT.Tout"
        ]
      },
      {
        "id": "weights_node",
        "label": "weights",
        "value": "{\"epa.HT.Tin\": 3.0}"
      }
    ],
    "connections": [
      {
        "source": {
          "node": "multivariate_mean",
          "port": "output"
        },
        "target": {
          "node": "output_node"
        }
      },
      {
        "source": {
          "node": "inputs_node"
        },
        "target": {
          "node": "multivariate_mean",
          "port": "inputs"
        }
      },
      {
        "source": {
          "node": "weights_node"
        },
        "target": {
          "node": "multivariate_mean",
          "port": "weights"
        }
      },
    ]
  }
}
```

The properties of the workflow creation document are as follows:

Property	Description

`name`	The workflow's human-friendly name.
`description`	The workflow's description.
`modelid`	The ID of the model that will be executed by the workflow.
`organisationid`	The ID of the Senaps organisation that will own the workflow.
`groupids`	The ID(s) of the Senaps groups that own the workflow within the organisation.
`runas`	The authorisation context for the workflow. Models within the workflow will run using the context provided and ensure secure access to the Senaps APIs. The `runas.roles` property must be a non-empty array containing 1 or more valid Senaps role ids.
`graph`	The workflow graph containing a list of "nodes" and "connections"
`graph.nodes`	An array of nodes that make up the workflow graph. (see next table following).
`graph.connections`	An array of connections that describe the links between nodes in a workflow (see next table following).

A "node" declares a reference to specific Senaps models, streams, grids, or text documents. A "connection" binds specific Senaps streams, grids, or text documents to one of the model's input or output ports. These documents have the following properties:

Nodes

Property	Description
`id`	The unique id for the node. This id is used when creating connections (See "Connections" below).
`label`	The user friendly label for display purposes.
`modelid`	**Model nodes only**. The ID of the Senaps model.
`streamid`	**Stream nodes only**. The ID of the Senaps stream.
`streamids`	**Multi-stream nodes only**. A list of Senaps stream IDs.
`value`	**Document nodes only**. The text document value.
`catalog`	**Grid nodes only**. The URL of the Thredds Data Server catalog.
`dataset`	**Grid nodes only**. The relative path of the gridded dataset within the Thredds Data Server.

Connections

Property	Description
`source`	The source node details that will be starting point for the connection.
`target`	The target node details that will be the end point for the connection.
`source.node`	The node id reference that exists in the `nodes` collection.
`source.port`	**Model nodes only**. The model's output port that will be used in the connection. Must match the type declared in the model's port.
`target.node`	The node id reference that exists in the `nodes` collection.

`target.port`	**Model nodes only**. The model's input port that will be used in the connection. Must match the type declared in the model's port.

If workflow creation is successful, the response body contains the new workflow definition, in the same format as the `GET /workflows/<workflow_id>` operation (see "Retrieving Workflows").

Retrieving Workflows

A list of the currently defined workflows can be retrieved by issuing a `GET` request to the API's `/workflows` endpoint.

This endpoint supports paginated requests using the `skip` and `limit` query parameters. Use `skip` to skip past a number of workflows in the overall list, and use `limit` to limit the number of workflows returned by the request. For example, if paginating with 20 worfklows per page, the third page can be retrieved by setting `skip=40` and `limit=20`. If omitted, `skip` defaults to zero, and `limit` defaults to 1000.

For example, to retrieve a page containing three workflows:

```
GET /api/analysis/workflows?skip=3&limit=3 HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base-64 encoded>>

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/hal+json

{
  "skip": 3,
  "limit": 3,
  "count": 3,
  "totalcount": 2335,
  "_links": {
  "next": {
    "href": "https://senaps.io/api/analysis/workflows/?skip=6&limit=3"
  },
  "previous": {
    "href": "https://senaps.io/api/analysis/workflows/?skip=0&limit=3"
  },
  "last": {
    "href": "https://senaps.io/api/analysis/workflows/?skip=2332&limit=3"
  },
  "self": {
    "href": "https://senaps.io/api/analysis/workflows/?skip=3&limit=3"
  },
  "first": {
    "href": "https://senaps.io/api/analysis/workflows/?skip=0&limit=3"
  }
  },
  "_embedded": {
  "workflows": [
    {
    "id": "d8f116b4-73de-49ee-91ab-32939be8f950",
    "name": "VAR predictor test",
    "description": "VAR predictor test with EPA data",
    "organisationid": "csiro",
    "groupids": [],
    "_links": {
      "self": {
      "href": "https://senaps.io/api/analysis/workflows/d8f116b4-73de-49ee-91ab-32939be8f950"
      }
    }
    },
    {
    "id": "709f0409-bbea-4ccf-a014-77c3bc1c72cf",
    "name": "Multivariate Mean workflow",
    "description": "New Multivariate Mean workflow",
    "organisationid": "csiro",
    "groupids": [],
    "_links": {
      "self": {
      "href": "https://senaps.io/api/analysis/workflows/709f0409-bbea-4ccf-a014-77c3bc1c72cf"
      }
    }
    },
    {
    "id": "e7cdc61e-eb39-4c3d-a783-85f937c6604d",
    "name": "Multivariate Mean workflow with weights",
    "description": "New Multivariate Mean workflow",
    "organisationid": "csiro",
    "groupids": [],
    "_links": {
      "self": {
      "href": "https://senaps.io/api/analysis/workflows/e7cdc61e-eb39-4c3d-a783-85f937c6604d"
      }
    }
    }
  ]
  }
}
```

The returned document consists of the following properties:

Property	Description
`skip`	The value of the `skip` query parameter.
`limit`	The value of the `limit` query parameter.
`count`	The number of workflows in the current "page" of results.
`totalcount`	The total number of workflows.
`_links.self.href`	The URL of the current "page" of results.
`_links.first.href`	The URL of the first "page" of results. Omitted if the current page is the first page.
`_links.last.href`	The URL of the last "page" of results. Omitted if the current page is the last page.
`_links.previous.href`	The URL of the previous "page" of results. Omitted if the current page is the first page.
`_links.next.href`	The URL of the next "page" of results. Omitted if the current page is the last page.
`_embedded.workflows`	A list of the workflows on the current "page" (see the next table following).

Each workflow in the `_embedded.workflows` list has the following properties:

Property	Description
`id`	The workflow's unique ID, used to refer to it elsewhere within the API.
`name`	The workflow's human-friendly name.
`description`	A description of the workflow.
`organisationid`	The ID of the Senaps organisation that created the workflow.
`groupids`	The ID(s) of the Senaps groups that own the workflow within the organisation.
`_links.self.href`	The workflow's unique API URL.

Note that some optional properties (in particular the `description`) may be omitted from the document if not specified for the workflow.

A workflow's unique URL consists of its ID appended to the `/workflows` endpoint URL. Performing a `GET` request on the workflow's URL will return a more detailed representation of the workflow.

For example, the following request retrieves the detailed representation of the workflow with ID `e7cdc61e-eb39-4c3d-a783-85f937c6604d` (a multivariate_mean model workflow):

```
GET /api/analysis/workflows/e7cdc61e-eb39-4c3d-a783-85f937c6604d HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base64 encoded>>

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/hal+json

{
  "id": "e7cdc61e-eb39-4c3d-a783-85f937c6604d",
  "name": "Multivariate Mean workflow with weights",
  "description": "New Multivariate Mean workflow",
```

```
"organisationid": "csiro",
"groupids": [],
"runas": { "roles": ["role1", "role2"] },
"_links": {
"self": {
  "href": "https://senaps.io/api/analysis/workflows/e7cdc61e-eb39-4c3d-a783-85f937c6604d"
}
},
"_embedded": {
    "graph": {
  "nodes": [
    {
      "id": "multivariate_mean",
      "label": "multivariate_mean",
      "operatorid": "multivariate_mean"
    },
    {
      "id": "output_node",
      "label": "output",
      "streamid": "epa.HT.Tave",
      "_embedded": {
        "datanode": {
          "streamid": "epa.HT.Tave",
          "type": "stream",
          "_links": {
              "stream": {
                  "href": "https://senaps.io/api/sensor/v2/streams/epa.HT.Tave"
              },
              "observations": {
                  "href": "https://senaps.io/api/sensor/v2/observations?streamid=epa.HT.Tave"
              }
          }
        }
      }
    },
    {
      "id": "inputs_node",
      "label": "inputs",
      "streamids": [
        "epa.HT.Tin",
        "epa.HT.Tout"
      ],
      "_embedded": {
        "datanode": {
          "streamids": [
          "epa.HT.Tin",
          "epa.HT.Tout"
        ],
          "type": "stream",
          "_links": {
            "stream": {
                "href": "https://senaps.io/api/sensor/v2/streams/epa.HT.Tave"
            },
            "observations": {
                "href": "https://senaps.io/api/sensor/v2/observations?streamid=epa.HT.Tave"
            }
          }
        }
      }
    },
    {
      "id": "weights_node",
      "label": "weights",
      "documentid": "bd97a39d-8874-4d96-870b-b7bc92655e69",
      "_embedded": {
          "datanode": {
              "value": "{\"epa.HT.Tin\": 3.0}",
              "documentid": "bd97a39d-8874-4d96-870b-b7bc92655e69",
              "organisationid": "csiro",
              "groupids": [],
              "type": "document",
              "_links": {
                  "self": {
                      "href": "https://senaps.io/api/analysis/documentnodes/bd97a39d-8874-4d96-870b-b7bc92655e69"
                  }
              }
          }
      }
```

```
      }
    ],
    "connections": [
      {
        "source": {
          "node": "multivariate_mean",
          "port": "output"
        },
        "target": {
          "node": "output_node"
        }
      },
      {
        "source": {
          "node": "inputs_node"
        },
        "target": {
          "node": "multivariate_mean",
          "port": "inputs"
        }
      },
      {
        "source": {
          "node": "weights_node"
        },
        "target": {
          "node": "multivariate_mean",
          "port": "weights"
        }
      },
    ]
  }
}
```

Each "data node" object describes the input or output bound to the corresponding model port. There are four kinds of data node, corresponding to the four port types: stream nodes, multistream nodes, document nodes and grid nodes. The following table outlines the properties of each:

Property	Description
`organisationid`	**Document nodes only**. The organisation that created the document node.
`groupids`	**Document nodes only**. The ID(s) of the Senaps groups that owns the document node within the organisation.
`streamid`	**Stream nodes only**. The ID of the stream to bind to the corresponding model port.
`streamids`	**Multi-stream nodes only**. The IDs of the streams to bind to the corresponding model port.
`value`	**Document nodes only**. The text value to bind to the corresponding model port.
`catalog`	**Grid nodes only**. The catalog URL to bind to the correspondint model port.
`dataset`	**Grid nodes only**. The dataset path to bind to the corresponding model port.
`_links.stream.href`	**Stream nodes only**. The Senaps URL for the stream's observations.
`_links.observations.href`	**Stream nodes only**. The Senaps observations URL for the stream bound to the model port.
`_links.self.href`	**Document nodes only**. The document node's unique URL.

Executing Workflows

Workflow execution is achieved by creating jobs (*see "Jobs"*). Making a `POST` request to the `/jobs` resource URL will issue a new "job request" which will be processed in the backgorund asynchronously. The resulting job resource URL can be used to query the status of a job. Making a 'GET' request to the `/jobs/{jobid}` endpoint will provide details on how job is progressing.

Special Operations

MAP

MAP is a special operation that takes an operator and applies it to all elements of a collection, potentially in parallel.

When sumbitting a workflow, an operator node can be 'mapped' by adding the `"type": "map"` property to the graph node definition.

For example, given a simple workflow definition (without MAP):

```
         +---------+
input -> | mymodel | -> output
         +---------+

// workflow graph json
"graph": {
  "nodes": [
    { "id": "mymodel", "operatorid": "mymodel" },
    { "id": "input", "value": "foo" },
    { "id": "output", "value": "" }],
  "connections": [
    {
      "source": { "node": "input" },
      "target": { "node": "mymodel", "port": "input" }
    },
    {
      "source": { "node": "mymodel", "port": "output" },
      "target": { "node": "output" }
    }
  ] // etc
```

The mapped version would be:

```
        // MAP(mymodel)

           +---------+
        -> | mymodel | ->
           +---------+
           +---------+
inputs  -> | mymodel | ->  outputs
           +---------+
           +---------+
        -> | mymodel | ->
           +---------+
            // etc...

// workflow graph json
"graph": {
    "nodes": [
      { "id": "mymodel", "type": "map", "operatorid": "mymodel"},
      { "id": "inputs",
        "collection": [
          { "value": "foo1" },
          { "value": "foo2" },
          { "value": "foo3" }]},
      { "id": "outputs",
        "collection": [
          { "value": "" },
          { "value": "" },
          { "value": "" }]}
    ],
    "connections": [
      {
        "source": { "node": "inputs" },
        "target": { "node": "mymodel", "port": "input" }
      },
      {
        "source": { "node": "mymodel", "port": "output" },
        "target": { "node": "outputs" }
      }
    ] // etc
```

Further Explanation:

Let's say `mymodel` was instead a `sum` operator that adds two streams together to produce a single output stream. The mapped version `MAP(sum)` would accept two stream collections pairing them element-wise to produce a single output stream collection.

`e.g. sum(A,B)=C with the 'Map' option becomes sum(A[], B[]) = C[]`

When the operator is run, internally the operations are mapped to the individual elements of the collections and executed potentially in parallel:

```
sum(A[0],B[0])=C[0]
sum(A[1],B[1])=C[1]
...
sum(A[n-1],B[n-1])=C[n-1]
// where n is the collection length
```

It is also possible to assign a constant value that is repeated for every execution:

```
sum(x,B[0])=C[0]
sum(x,B[1])=C[1]
...
// where x is a stream node
```

Considerations:

- The length of a connected collection determines the number of executions to perform.
- All collection inputs and outputs must be of equal length.
- A single value can be supplied as a constant value which will be repeated for every execution.
- A simple operator that has collection ports cannot be 'mapped'. Mapping collections of collections is not currently supported.
- Any connecting downstream operators will be triggered once ALL mapped operations have completed successfully.

Jobs

Jobs can be submitted to trigger workflow execution. To create a job, a `POST` request is made to the `/jobs` endpoint which must include a `workflowid` to execute and optionally a `debug` flag. At each stage of execution, a job is updated with the current status, execution history, debugging information such as log and error messages if enabled, and other useful statistics such as execution time for example.

Submitting a Job

Below is an example of a HTTP request and response when creating a job:

```
POST /api/analysis/jobs HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base-64 encoded>>

{
  "workflowid": "e7cdc61e-eb39-4c3d-a783-85f937c6604d",
  "debug": true
}

HTTP/1.1 202 Accepted
Connection: keep-alive
Content-Type: application/hal+json

{
    "id": "2ce588b1-f543-4b70-923d-e79de5f26110",
    "status": "Queued",
    "timestamp": "2017-10-18T11:12:53.042Z",
    "debug": true,
    "history": [
        {
            "status": "Created",
            "timestamp": "2017-10-18T11:12:53.042Z"
        },
        {
            "status": "Queued",
            "timestamp": "2017-10-18T11:12:53.042Z"
        }
    ],
    "workflowid": "e7cdc61e-eb39-4c3d-a783-85f937c6604d",
    "organisationid": "csiro",
    "groupids": [
    ],
    "_links": {
        "workflow": {
            "href": "https://senaps.io/api/analysis/workflows/e7cdc61e-eb39-4c3d-a783-85f937c6604d"
        },
        "self": {
            "href": "https://senaps.io/api/analysis/jobs/2ce588b1-f543-4b70-923d-e79de5f26110"
        }
    }
}
```

Polling a Job

While the HTTP status code returned in a job response is `202 Accepted`, polling can be achieved by making a `GET` request to the `_links.self.href` property value contained in the HAL resposne until a `200 OK` HTTP status code is returned.

- A `202 Accepted` HTTP status code signifies that jobs is not yet complete and can be polled periodically by issuing a `GET` request on the `/jobs/{jobid}` to recieve updates.
- A `200 OK` HTTP status code signifies that the job is complete and will remain unchanged from that point forward.

Below is an example of a completed job:

```
GET /api/analysis/workflows/e7cdc61e-eb39-4c3d-a783-85f937c6604d HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base64 encoded>>

HTTP/1.1 200 OK
Connection: keep-alive
Content-Type: application/hal+json

{
    "id": "2ce588b1-f543-4b70-923d-e79de5f26110",
    "status": "Success",
    "timestamp": "2017-10-18T11:13:12.512Z",
    "debug": true,
    "history": [
        {
            "status": "Created",
            "timestamp": "2017-10-18T05:12:53.042Z"
        },
        {
            "status": "Queued",
            "timestamp": "2017-10-18T05:12:53.042Z"
        },
        {
            "status": "Processing",
            "timestamp": "2017-10-18T05:12:53.804Z"
        },
        {
            "status": "Dispatched",
            "timestamp": "2017-10-18T05:12:53.806Z"
        },
        {
            "status": "Success",
            "timestamp": "2017-10-18T05:13:12.512Z"
        }
    ],
    "workflowid": "e7cdc61e-eb39-4c3d-a783-85f937c6604d",
    "organisationid": "csiro",
    "groupids": [],
    "_links": {
        "workflow": {
            "href": "https://senaps.io/api/analysis/workflows/e7cdc61e-eb39-4c3d-a783-85f937c6604d"
        },
        "self": {
            "href": "https://senaps.io/api/analysis/jobs/2ce588b1-f543-4b70-923d-e79de5f26110"
        }
    },
    "_embedded": {
        "results": {
            "statistics": {
                "starttime": "2017-10-18T05:12:53.819Z",
                "elapsedtime": "PT18.529S",
                "status": "Success",
                "endtime": "2017-10-18T05:13:12.348Z",
                "errors": [],
                "log": [
                    {
                        "message": "Calling implementation method for model multivariate_mean...",
                        "timestamp": "2017-10-18T05:12:54.548Z",
                        "level": "DEBUG",
                        "file": "model.py",
                        "line": 101,
                        "logger": "JobProcess"
                    }
                ]
            }
        }
    }
}
```

The following table describes the various job states.

Status	Description
Created	The job has been created but not yet queued
Queued	The job has been queued
Processing	The job has been pulled from the queue and is being processed
Dispatched	The job has been dispatched to the execution environment and is running
Error	There was an error
Success	The job completed successfully
Failed	The job failed due to an error or some other reason

Document Nodes

Document nodes are the storage mechanism for textual data that are used primarily as parameter inputs and outputs for models.

Stored globally, the same document can be referenced by multiple workflows allowing for common input parameters to be reused and centralise variable maintenance. A document node can be created inline during workflow creation by specifying a `value` property, or can be referenced using the `documentid` when declaring a node in a workflow graph node array.

It's worth noting that in the specific case of output document nodes, the value contained within represents the value output by the model. In all other cases (all stream and multistream data nodes, all input data nodes) the data node is unchanged following workflow execution.

Future versions plan support richer document formats with the ability to include supporting schemas for document validation.

Schedules

Schedules can be created to schedule a workflow to execute at certain times by making a `POST` request to the `/schedules` endpoint. A "schedule request" must contain a valid cron expression (https://en.wikipedia.org/wiki/Cron) , a `workflowid`, and a `name` for the schedule.

Schedules can be toggled between active and inactive states by setting the `active` flag on a schedule by issuing a `PUT` request to the `/schedules` endpoint.

Compute Profiles

Models are provided a guaranteed CPU and RAM allocation when run in the Senaps Analysis Service. At present, the default allocation consists of ¼ of a CPU (time-shared with other processes) and a 256 MB RAM allocation. Depending on load, models may exceed these limits if further resources are available, although this cannot be guaranteed.

In cases where a higher CPU and/or RAM allocation is required, an optional "compute profile" may declared either in the definition of a model (see the Model Developer's Guides), or when declaring a model node in a workflow. These profiles consist of a specified CPU and RAM allocation, which is then applied when the model is executed. To retrieve a full list of available profiles, see the API's `/profiles` endpoint.

To declare a compute profile in a workflow, list its ID in the model's node declaration (see the `profileid` property):

```
POST /api/analysis/workflows HTTP/1.1
Host: senaps.io
Content-Type: application/json
Authorization: Basic <<username:password base-64 encoded>>

{
  "name": "High-Memory Multivariate Mean workflow",
  "description": "Multivariate Mean workflow with increased CPU and RAM allocation.",
  "organisationid": "csiro",
  "groupids": [],
  "runas": { "roles": [ "my_example_role" ]}
  "graph": {
    "nodes": [
      {
        "id": "multivariate_mean",
        "label": "multivariate_mean",
        "operatorid": "multivariate_mean",
        "profileid": "XL"
      },
      {
        "id": "output_node",
        "label": "output",
        "streamid": "epa.HT.Tave"
      },
      {
        "id": "inputs_node",
        "label": "inputs",
        "streamids": [
          "epa.HT.Tin",
          "epa.HT.Tout"
        ]
      },
      {
        "id": "weights_node",
        "label": "weights",
        "value": "{\"epa.HT.Tin\": 3.0}"
      }
    ],
    "connections": [
      {
        "source": {
          "node": "multivariate_mean",
          "port": "output"
        },
        "target": {
          "node": "output_node"
        }
      },
      {
        "source": {
          "node": "inputs_node"
        },
        "target": {
          "node": "multivariate_mean",
          "port": "inputs"
        }
      },
      {
        "source": {
          "node": "weights_node"
        },
        "target": {
          "node": "multivariate_mean",
          "port": "weights"
        }
      },
    ]
  }
}
```

Compute profiles declared in this manner override both the default profile **and** any profile declared in the model itself.