STScI | SPACE TELESCOPE SCIENCE INSTITUTE

EXPANDING THE FRONTIERS OF SPACE ASTRONOMY

UWS through OpenAPI

Joshua Fraustro

November 15th, 2024

What were the goals?

What would a UWS 1.1 OpenAPI specification look like?

Why UWS?

- A sufficiently complicated example of a web service pattern that is already very RESTful in its design.
- No part of it requires describing / modeling data formats. (VOTables)

Demonstrate:

- Paths, operations, parameters, and protocol models could be adequately represented.
- Version changes, iterative and large updates, were easy to create and work against.
- We could take advantage of modern tooling that uses OpenAPI standards.

Problems? What problems?

Some of the problems & HTTP/REST issues in UWS have been pointed out in previous Interops.

Some are small and simple to fix, some are definitely breaking

- Case-insensitive query parameters
- 303 HTTP status codes for successful operations (creating Jobs)
- POST operations for updating Job parameters
- Difficult to describe nuances of the XML schema
- Empty response bodies for certain Job parameters
- Unclear which parameters can be updated with POSTs to their endpoint.

See: P3T Sydney 2024 & DAL Bologna 2023

We listened to feedback.

I presented a version of UWS in OpenAPI in Sydney & virtually.

- This version had the OpenAPI doc + all of the “fixes”.

Generally:

- People liked the idea of adding this kind of documentation.
- Felt it added an extra option for client & service developers.

But also concerns:

- Too much, too fast, breaking changes, interoperability concerns

How do we bridge the gap from here, to there?

Describing UWS in 3 steps

As an exercise, create 3 versions of the spec, showing we can iterate:

“ As-Is ”	“ Refinement ”	“ What-If? ”
• Describe the current UWS 1.1 standard as closely as possible • Only make changes that are not otherwise possible to avoid.	• Solve a few problems with small changes • Non-breaking / as easy for client developers as possible	• Solve the highlighted problems, knowing things will break • Replace XSD definitions with OpenAPI schema models

UWS OpenAPI v1.1 - “As-Is”

Version 1:

A mostly straight-forward description of UWS 1.1 into an OpenAPI specification.

Positives:

- Fairly simple to do, UWS is already very RESTful in its design.
- Paths, operations easy to document — no conflict with OpenAPI specs.
- Avoid trying to model the XML request / responses, only point to current XSD.

Notes:

- OpenAPI 3.0 has XML support, but too basic for the UWS schema.
- Case-insensitive query parameters are not describable.
- Redocly linting immediately noticed HTTP status code problem.

```yaml
paths:
  /: ⋯
  /{job-id}:
    parameters: ⋯
    get: ⋯
    post: ⋯
    delete: ⋯
  /{job-id}/phase:
    parameters: ⋯
    get: ⋯
    post: ⋯
  /{job-id}/executiondur
    parameters: ⋯
    get: ⋯
    post: ⋯
```

```yaml
/{job-id}:
  parameters: ⋯
  get:
    Scan | Try it | Audit
    operationId: getJobSummary
    summary: 'Returns the job summary'
    parameters: ⋯
    responses:
      '200':
        description: Success
        content:
          application/xml:
            schema:
              $ref: '#/components/schemas/Job'
      '403':
        $ref: '#/components/responses/Forbidden'
      '404':
        $ref: '#/components/responses/JobNotFound
```

```yaml
JobSummary:
  type: object
  description: |
    The complete representation of the sta
  title: jobSummary
  required: [jobId]
  properties:
    jobId:
      type: string
      description: |
        The identifier for the job
      example: 'HSC_XYZ_123'
    runId:
      type: string
      maxItems: 1
      description: |⋯
      example: 'JWST-1234'
    ownerId:
      type: string
      nullable: true
      description: |⋯
      example: 'Noirlab/John.Smith'
    phase:
      $ref: '#/components/schemas/Executio
```

“As-Is” OpenAPI Document

- Examples here of:
  - Path parameters and operations
  - Response enumeration
  - Request & response models

Github Link:

https://github.com/jwfraustro/PTTT/tree/uws-basic

UWS OpenAPI v1.2 - “Refinement”

Version 2: Small changes to fix design issues without greatly changing the spec.

- Minimal to implement for services, clients.

  An example of how it’s **easy to see changes** in the OpenAPI document.

Change	Example	Breaking?
All query parameters shall be lowercase or camelCase (for multi-word params)	PHASE -> phase OtherParam -> otherParam	No (in the context of TAP). DALI requires arbitrary casing.
HTTP status codes for GET’s to empty parameters should indicate as such	GET /quote 204, No-Content or 200, null	Yes, but a simple change.
303 Redirects (for POST’s, etc.) changed to their appropriate status codes	POST /jobs should return 200 OK	Yes, but a simple change.

```yaml
paths:
  /:
    get:
      Scan | Try it | Audit
      operationId: getJobList
      summary: Returns the list of UWS jobs
      parameters:
        - name: phase …
        - name: after …
        - name: last …
      responses: …
    post:
      Scan | Try it | Audit
      operationId: postCreateJob
      summary: 'Submits a job'
      requestBody: …
      responses:
        '200': …
        '403': …
        '404': …
  /{job-id}: …
```

```yaml
operationId: getJobList
summary: Returns the list of UWS jobs
parameters:
  - name: PHASE
  - name: phase      You, 8 m
    in: query
    description: 'Execution p
    schema:
      $ref: '#/components/sch
  - name: AFTER
  - name: after
    in: query
    description: 'Return jobs
    schema:
      type: string
      format: date-time
  - name: LAST
  - name: last
    in: query
    description: 'Return only
    schema:
```

```yaml
/{job-id}/destruction:
  parameters:
    - $ref: '#/components/parameters/job-id'
  get:
    operationId: getJobDestruction
    summary: 'Returns the job destruction time'
    responses:
      '200':
        description: Success
        content:
          text/plain:
            schema:
              type: string
              format: date-time
      '204':
        description: 'No destruction time set'
      '403':
```

```yaml
                enum:
                  - "RUN"
                  - "ABORT"
                  - "SUSPEND"
                  - "ARCHIVE"
    responses:
      '303':
      '200':
        description: "Success"
        $ref: '#/components/responses/JobSummaryRedirect'
        $ref: '#/components/responses/JobSummaryResponse'
```

“Refinement” OpenAPI Document

- New OpenAPI document on the left
- Git diffs between the two versions are easy to see and understand.

Github Link:

https://github.com/jwfraustro/PTTT/tree/uws-improved

UWS OpenAPI vX - “What-if?”

Version X: The version I presented in Sydney.

Changes:

- All of the previous changes
- Request / response job messages are fully JSONSchema describable.
  - Means we get more varied native encoding formats.
- Job creation by POST’ing the document — no form-encoded key/values

Is it breaking?

- Well, yes.

Working on a prototype implementation at MAST w/ FastAPI client libraries.

```yaml
post:
  operationId: postCreateJob
  tags: [UWS]
  summary: 'Submits a job'
  requestBody:
    description: 'Job parameters'
    required: true
    content:
      application/x-www-form-urlencoded:
      application/json:
        schema:
          type: object
          properties:
            # Examples for TAP implementation
            QUERY:
              type: string
              description: 'The query to be performed'
              example: 'SELECT * FROM TAP_SCHEMA.tables'
            LANG:
              type: string
              description: 'The language in which the query should be performed'
              example: 'ADQL'
          additionalProperties: true
          $ref: '#/components/schemas/Parameters'
  responses:
```

“What-If” OpenAPI Document

- Removal of form-urlencoded.
- Simple to represent change, just point the request body at the ‘Parameters’ object

Github Link:

https://github.com/jwfraustro/PTTT/tree/UWS-MAST

What's the point?

Demonstrations here are:

Not about how easy the changes are, but **how easy they are to document.**

Show that:

- We can take iterative steps towards what was proposed.
- Iteration is clearly shown through these documents.
- We have a version that can be deployed "tomorrow".
- We understand what might break at each step.

Going Forward

What's next?

TAP 1.2 will be implementing an OpenAPI spec that needs UWS.

- Take a look at the "As-Is" version of the OpenAPI spec.
  - No functional changes in the standard.
  - Needs review from more eyes (than mine!)
- Think about how we would integrate OpenAPI docs with our current document publishing pipelines.
- Keep pushing forward with JSON-compatible implementation / libraries to understand what the future looks like.

Bonus: PetStore IVOA Spec

IVOA-style standards document for the classic PetStore API example:

2.2.3.1. Getting the Pet List

The list of pets available in the Pet Store may be retrieved by sending a GET request to the endpoint /pets. In this case, the query parameters that may be included in the request are LIMIT, which restricts the number of pet items returned, and STATUS, which allows the client to specify a filter based on the current availability of the pets. The status parameter accepts the values "AVAILABLE", "PENDING", or "SOLD". The response to this request will contain a JSON array of pet objects, each representing a distinct pet record in the system.

Upon successful retrieval, the response code will be 200 "OK", and the response body will contain a JSON representation of the pets matching the query parameters, if any. The server may also respond with a 400 "Bad Request" status code if the request parameters are invalid (for example, if limit is non-numeric or negative).

Bonus: PetStore IVOA Spec

Can you diff… the LaTeX?

```
2.2.3.1. Getting the Pet List

The list of pets available in the Pet Store may be retrieved by sending a GET request to the endpoint /pets. In this case, the query parameters that may be included in the request are LIMIT, which restricts the number of pet items returned, and STATUS, which allows the client to specify a filter based on the current availability of the pets. The status parameter accepts the values "AVAILABLE", "PENDING", or "SOLD". The response to this request will contain a JSON array of pet objects, each representing a distinct pet record in the system.

Upon successful retrieval, the response code will be 200 "OK", and the response body will contain a JSON representation of the pets matching the query parameters, if any. The server may also respond with a 400 "Bad Request" status code if the request parameters are invalid (for example, if limit is non-numeric or negative).
```

```
2.2.3.1. Getting the Pet List

A client may access the list of pets in the Pet Store by initiating a GET request to /pets. Clients can include optional query parameters in the request to influence the returned data. These parameters include QUANTITY, which specifies the desired count of pet items in the response, and STATE, which allows clients to specify a filter based on the current state of the pets, accepting values such as "ACTIVE", "RESERVED", or "SOLD". The server's response will include a JSON-formatted array of objects, each representing a pet with various attributes.

On successful execution of the request, the server will return a status code of 200 "OK", with a JSON body representing the list of pets that meet the specified criteria. The server may return a 400 "Bad Request" if it detects invalid parameter values, such as a non-numeric quantity or unrecognized state value.
```

Bonus: PetStore IVOA Spec

Can you diff… the OpenAPI spec?

```
/pets:
  get:
    tags:
      - pet
    summary: List all pets
    description: Returns all pets from the store
    operationId: listPets
    parameters:
      - name: limit
        in: query
        description: How many pets to return at one time (max 100)
        required: false
        schema:
          type: integer
          format: int32
      - name: status
        in: query
        description: Filter pets by status
        required: false
        schema:
          type: array
          items:
            type: string
            enum:
              - available
              - pending
              - sold
    responses:
```

```
/pets:
  get:
    tags:
      - pet
    summary: List all pets
    description: Returns all pets from the store
    operationId: listPets
    parameters:
      - name: quantity
        in: query
        description: How many pets to return at one time (max 100)
        required: false
        schema:
          type: integer
          format: int32
      - name: state
        in: query
        description: Filter pets by status
        required: false
        schema:
          type: array
          items:
            type: string
            enum:
              - active
              - reserved
              - sold
    responses:
```