OL® Connect

REST API Cookbook with Working Examples

Version 2023.1

Objectif Lune

REST API Cookbook with Working Examples
Version 2023.1
Last Revision: 5/13/2023

Upland Software, Inc

Table of Contents

Welcome to the PReS Connect REST API Cook-book

This guide is aimed at technically experienced users who wish to learn and use the REST API available in **PReS Connect** version 2023.1.

The PReS Connect REST API consists of many services that expose access to a number of areas including workflow, data entity management and file store operations.

These services can be used to perform various interactions with the PReS Connect server such as:

- Upload and manage data files, data mapping configurations and templates in the file store
- Create, manage and find data entities internal to the PReS Connect server
- Create and monitor processing operations within the workflow

The REST API also supports added security to restrict unauthorized access to the services.

For information about the server configuration, see the PlanetPress Connect Help.

This guide is broken down into three sections:

- Technical Overview – Overview of the concepts and structures used in PReS Connect and the REST API
- Working Examples – Working examples of the PReS Connect REST API in action (HTML5 & JavaScript/jQuery)
- REST API Reference – A complete reference to the PReS Connect REST API and services

It is recommended that the technical overview section be read first, followed by the working examples, using the REST API reference for greater detail on implementing any specific example.

Technical Overview

This section provides an overview of the concepts and structures used within PReS Connect and the REST API.

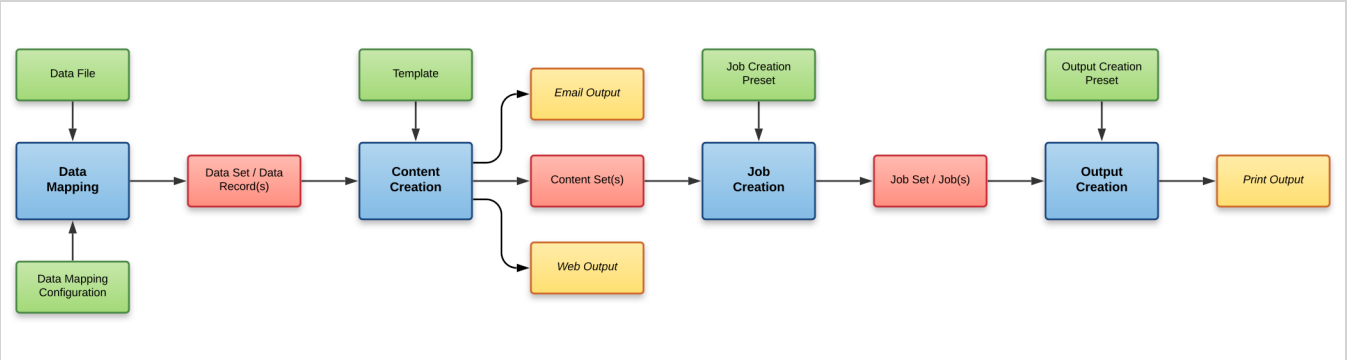
- Workflow & Workflow Processes
- Input Files
- Data Entities
- Workflow Operations
- JSON Structures

Workflow & Workflow Processes

The primary workflow in PReS Connect consists of four major processes that each require a number of inputs, and once executed, produce a particular form of output. These processes are: data mapping, content creation, job creation and output creation.

There is an additional workflow process, named All-In-One, which embodies all four major workflow processes in a singular process.

The following diagram illustrates the primary workflow in PReS Connect:



Typically an individual workflow process (shown above in *blue*) will take one or more input files as input (shown above in *green*), and will produce either intermediary output in the form of a data entity (shown above in *red*), or final output in the form of print, web, or email based content (depending on the context of the content produced) (shown above in *yellow*).

Input files to a workflow process include files such as data files, data mapping configurations and templates. In most cases an input file needs to be uploaded to the server file store before it can be used in a workflow process. A file that has been uploaded to the file store is known as a managed file, and managed files can be referenced via a unique identifier or name.

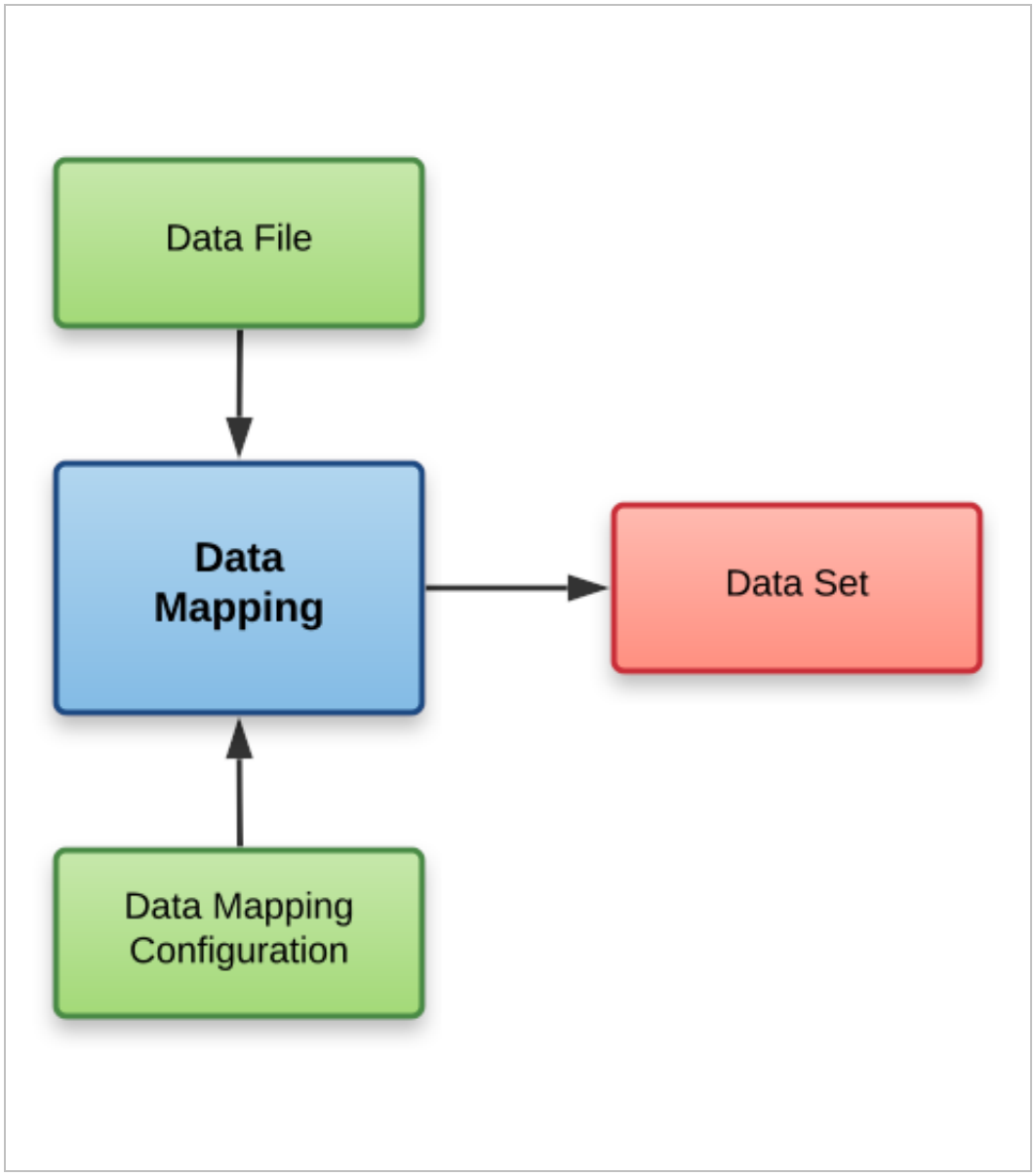
A data entity is simply a structured data artefact, produced as a result of an instance of a workflow process known as a workflow operation. Data entities are stored internally to the server and can also be referenced via a unique identifier.

Where a certain process depends on the output of the process before it, the data entity or entities produced by the earlier process are used as an input to that process.

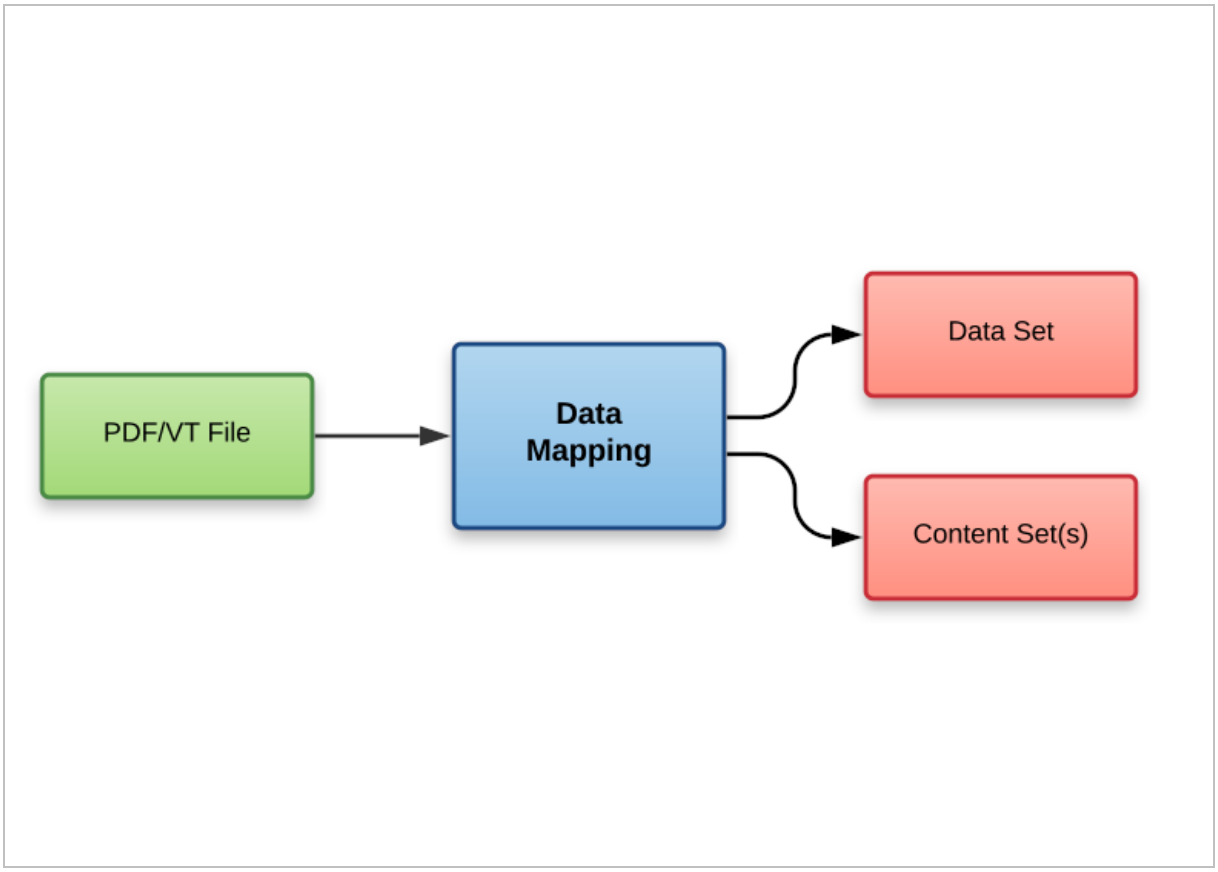
Data Mapping

The data mapping process involves taking a data file or source, applying a data mapping configuration to it, and producing a structured set of data or data records (a data set). This process can also produce a data set or content set from a PDF/VT file using its internal meta data instead of a data mapping configuration.

The following diagram illustrates the default workflow for the data mapping process:



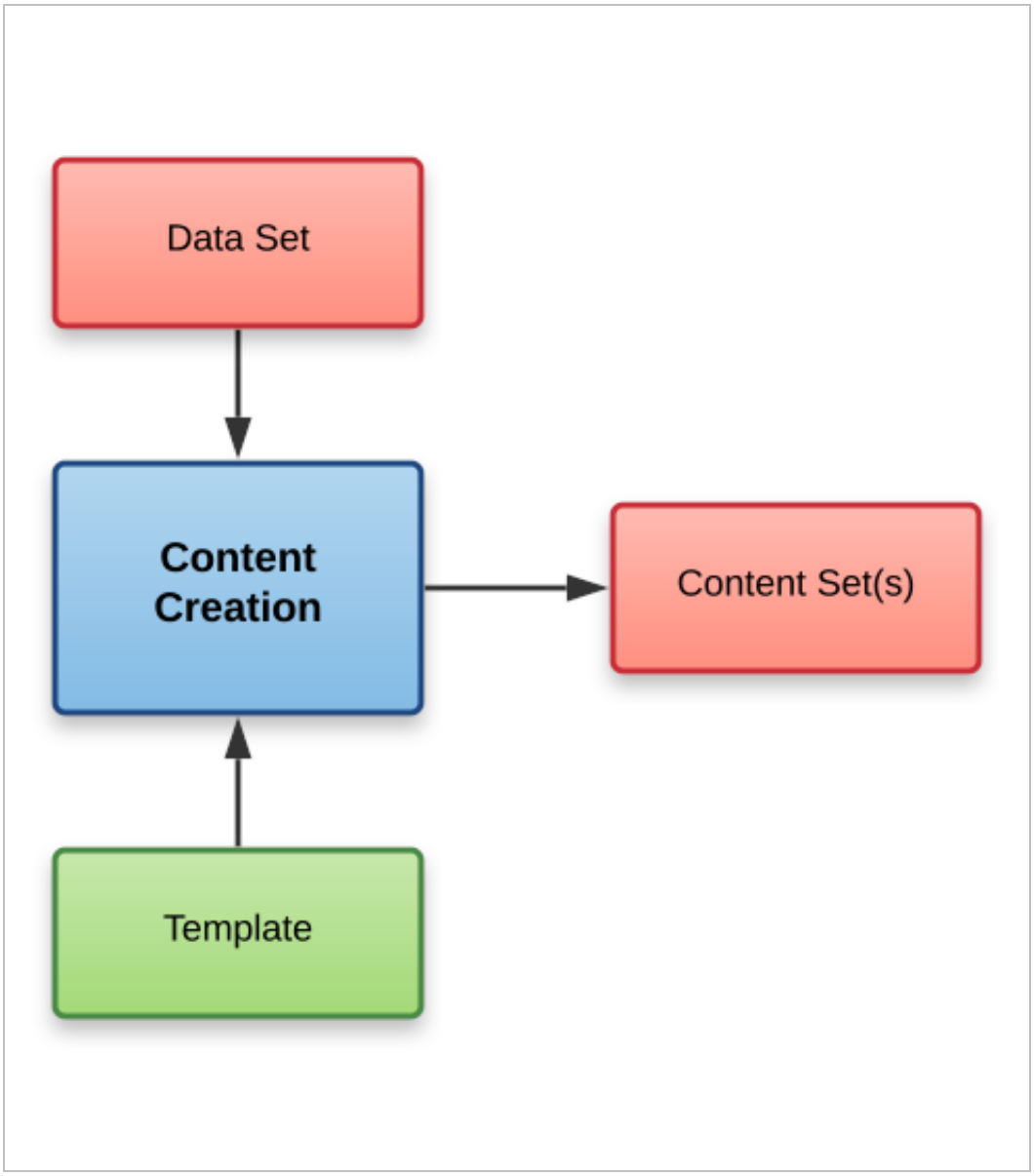
The following diagram illustrates the alternative workflow for the data mapping process when using PDF/VT data files specifically:



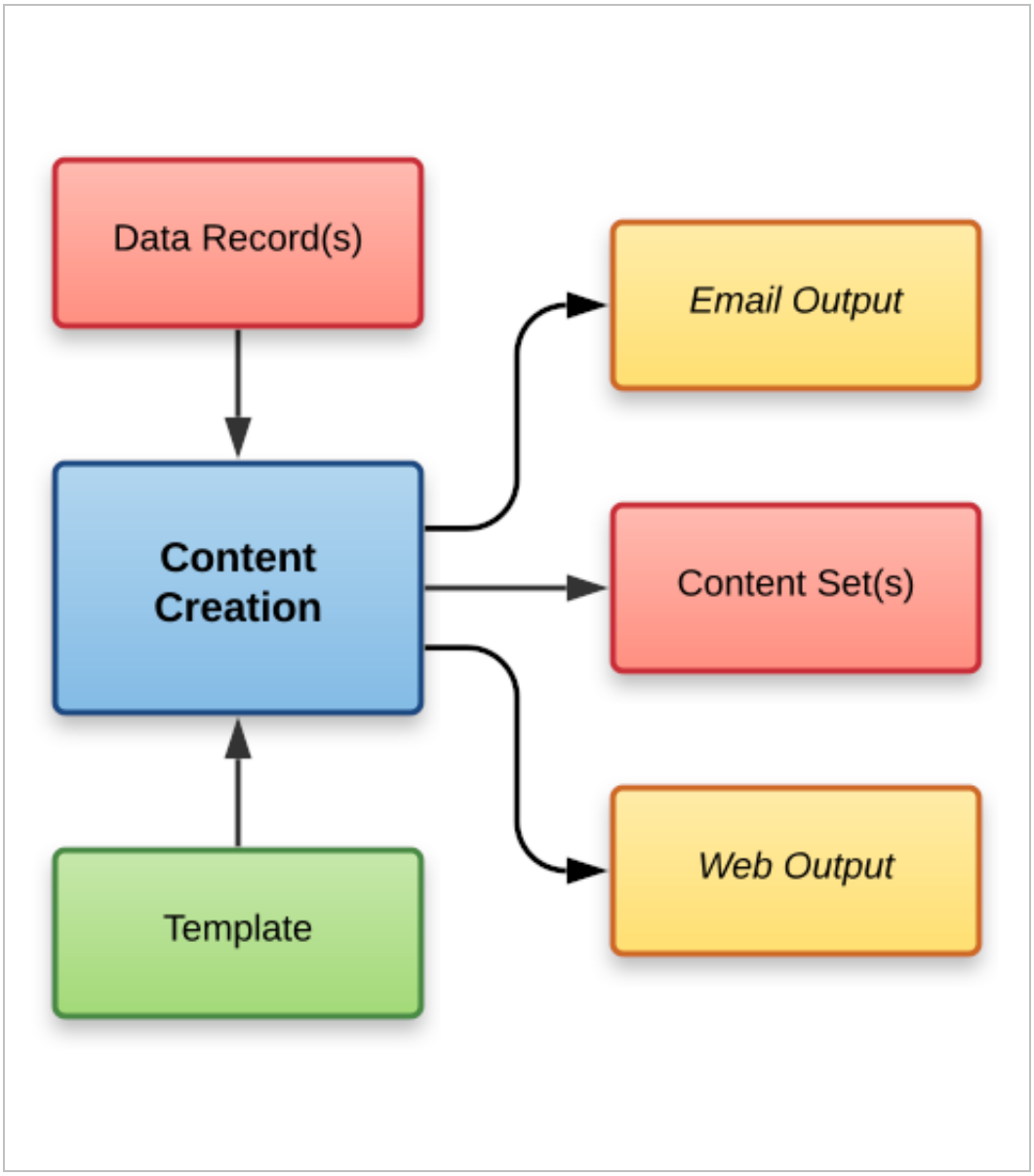
Content Creation

The content creation process involves taking either a data set or one or more data records (from a data set), combining them with a suitable template, and producing one or more sets of content (content sets). If the content is for the Email or Web context then output can be produced at this stage.

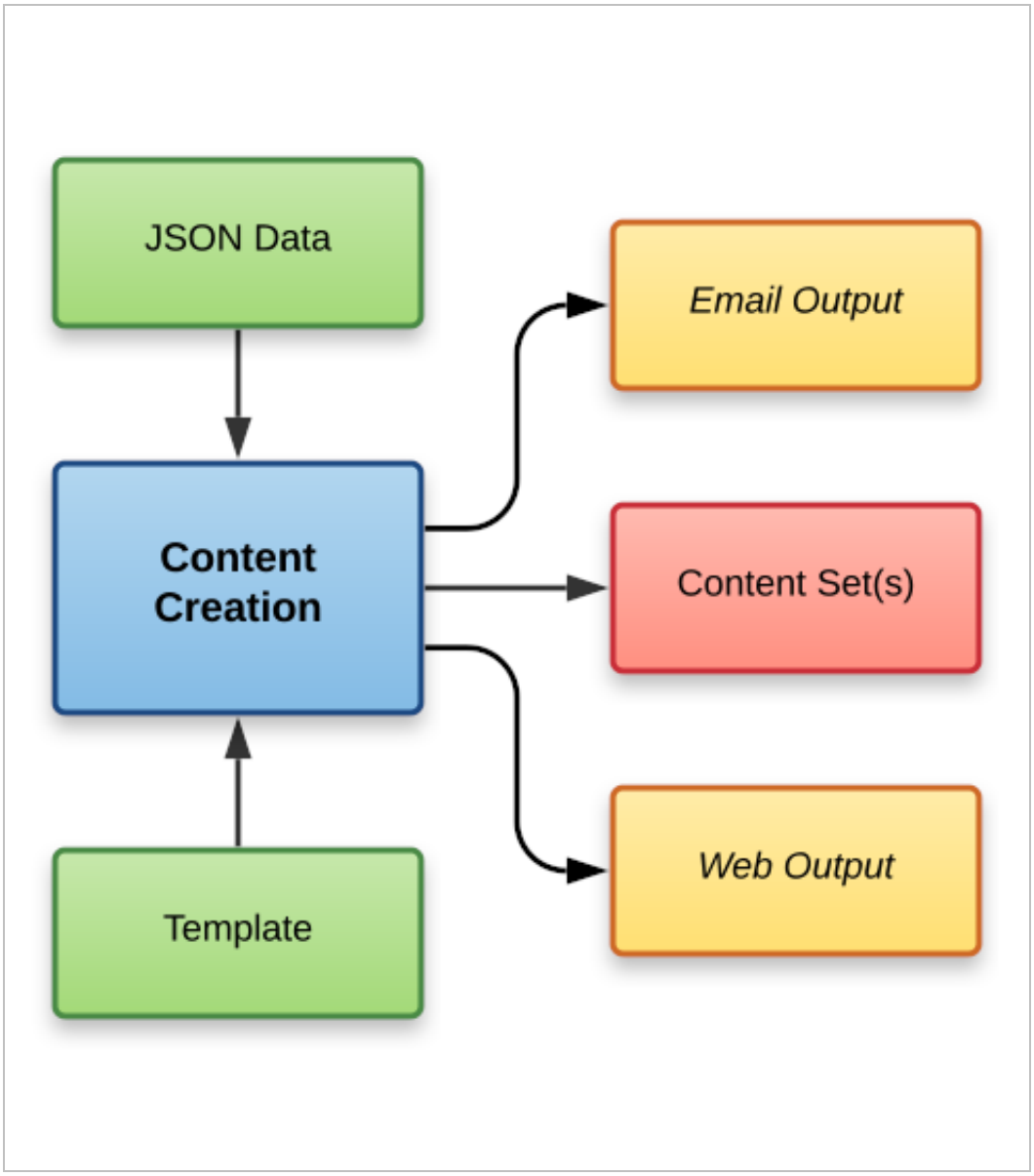
The following diagram illustrates the workflow for content creation by **Data Set**:



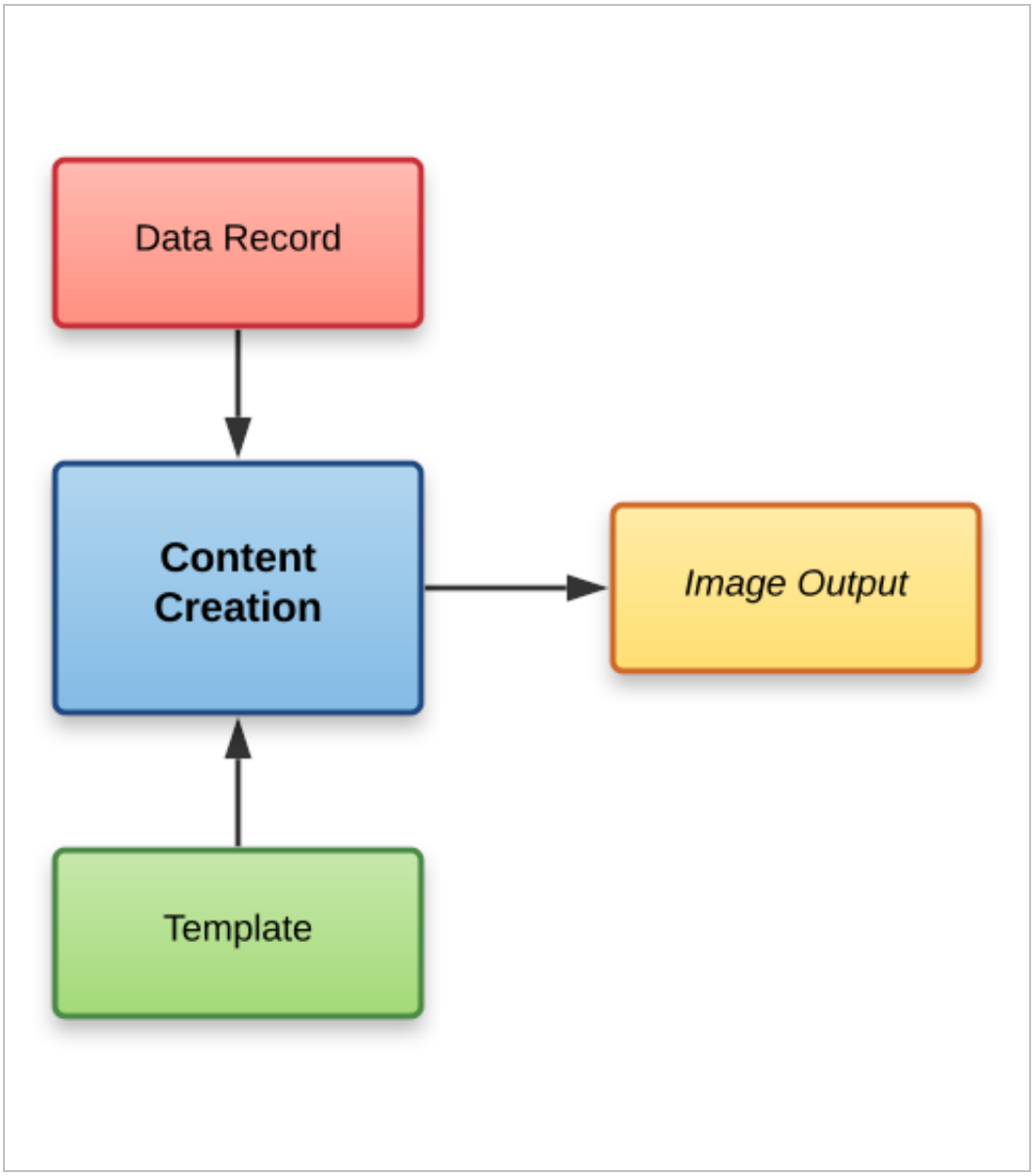
The following diagram illustrates the workflow for content creation by **Data Record**:



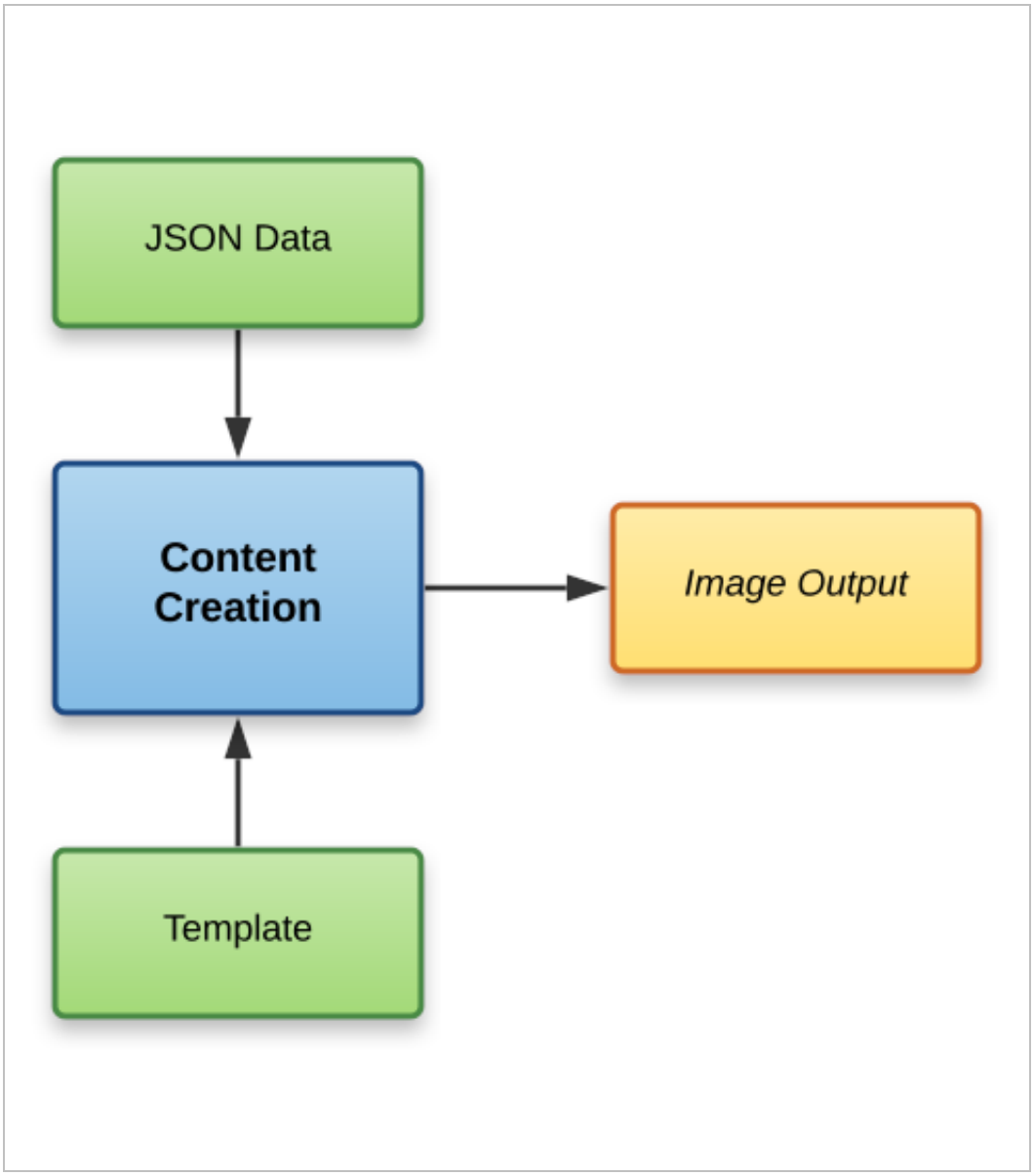
The following diagram illustrates the workflow for content creation by **JSON Data**:



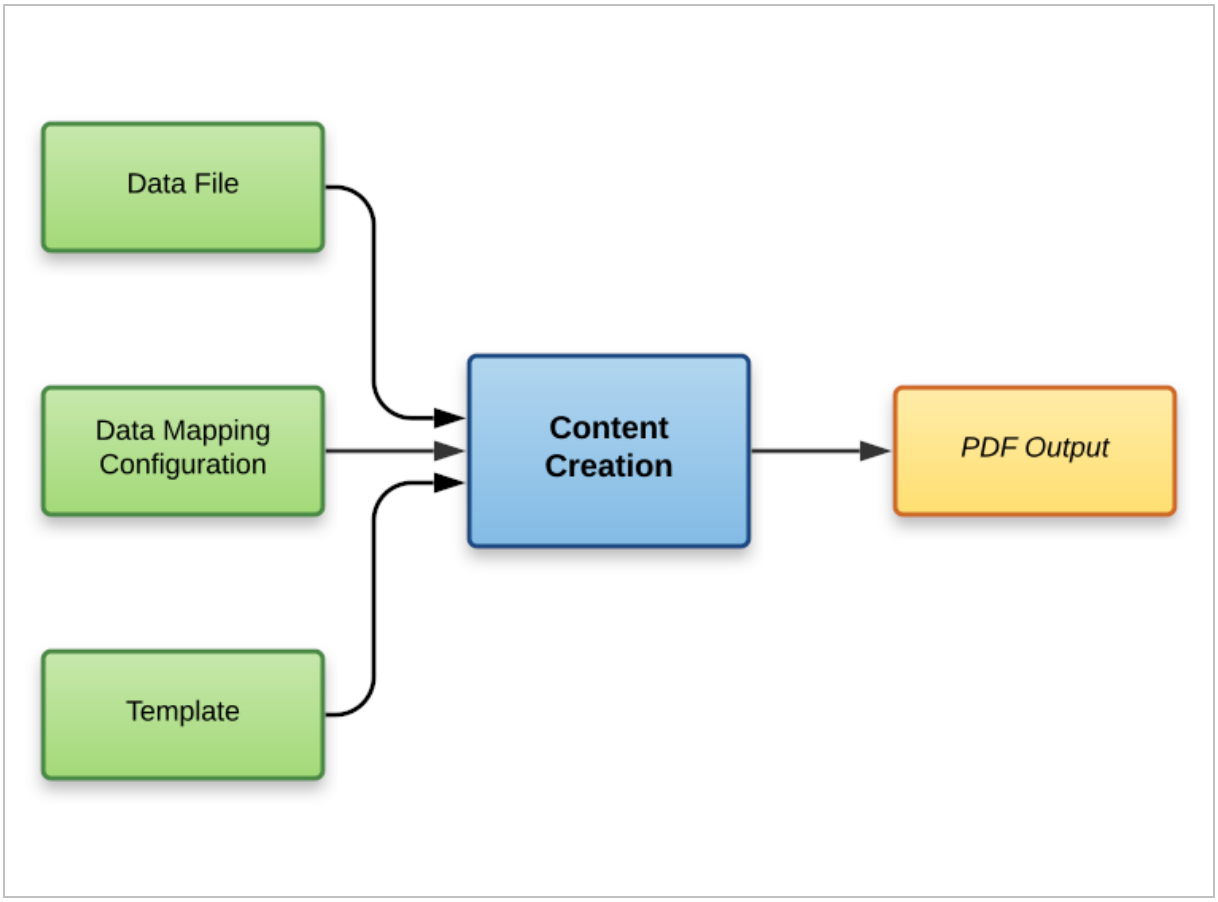
The following diagram illustrates the workflow for content creation of **Preview Image**, by **Data Record**:



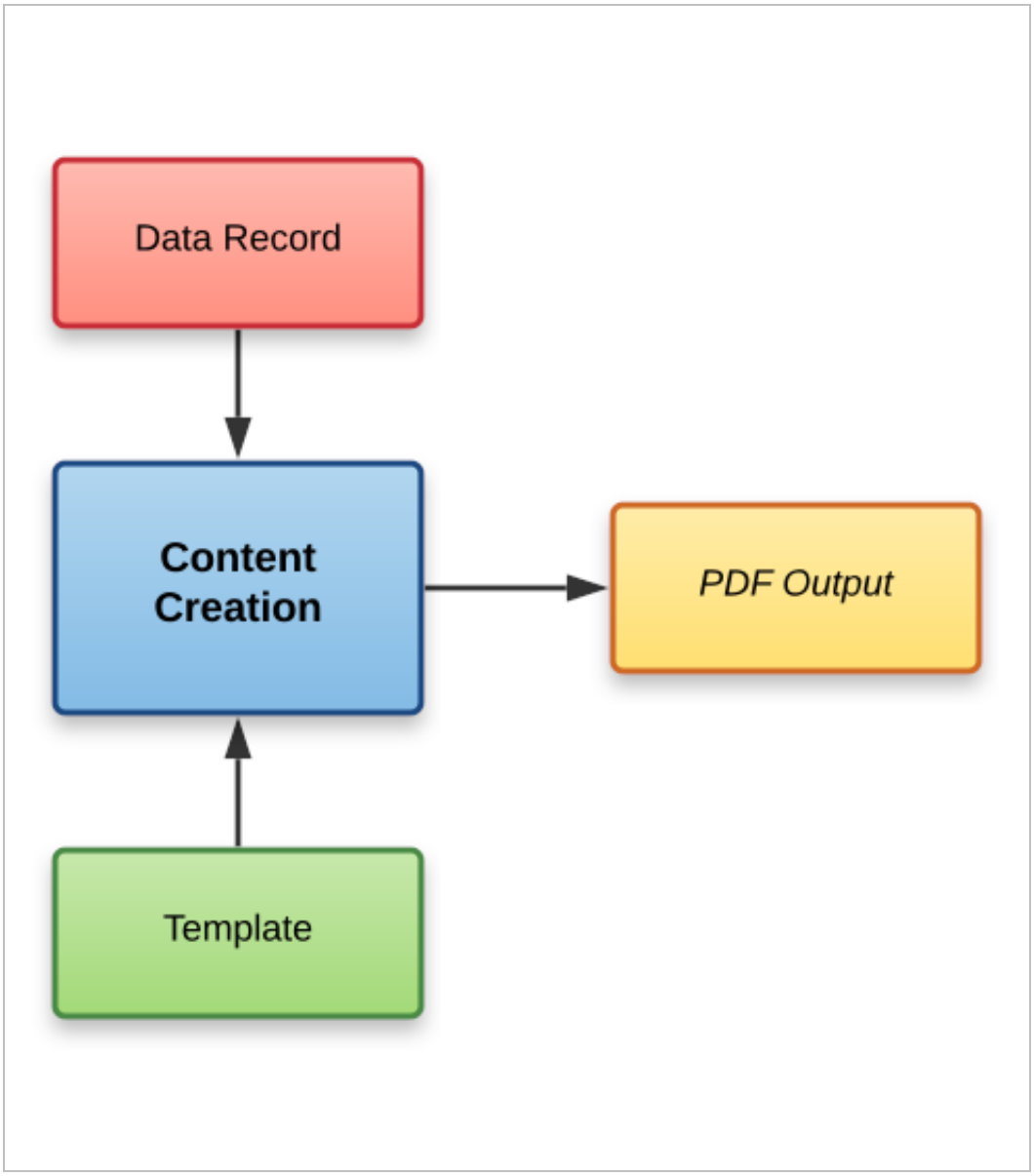
The following diagram illustrates the workflow for content creation of **Preview Image**, by **JSON Data**:



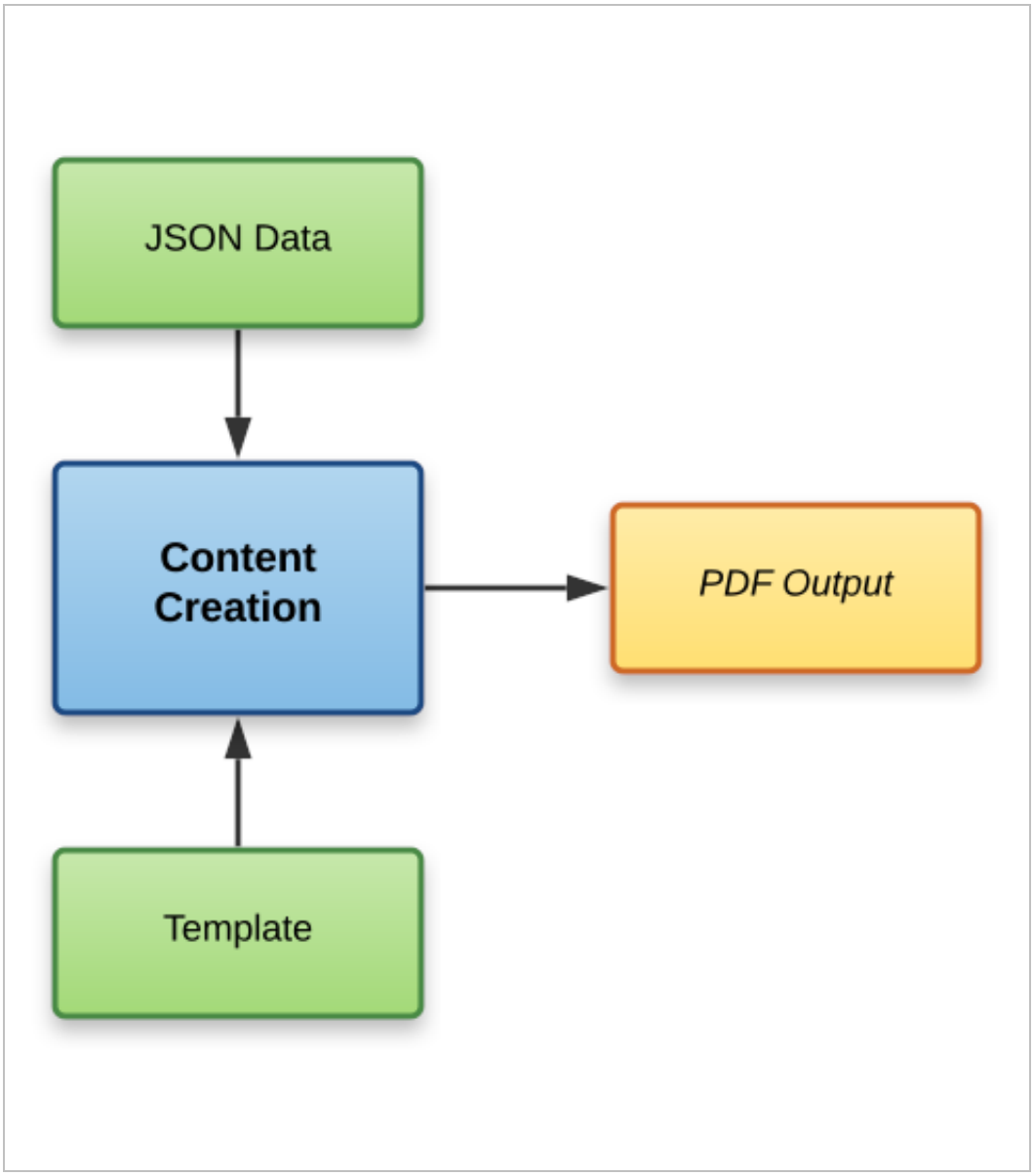
The following diagram illustrates the workflow for content creation of **Preview PDF**, by **Data Mapping**:



The following diagram illustrates the workflow for content creation of **Preview PDF**, by **Data Record**:



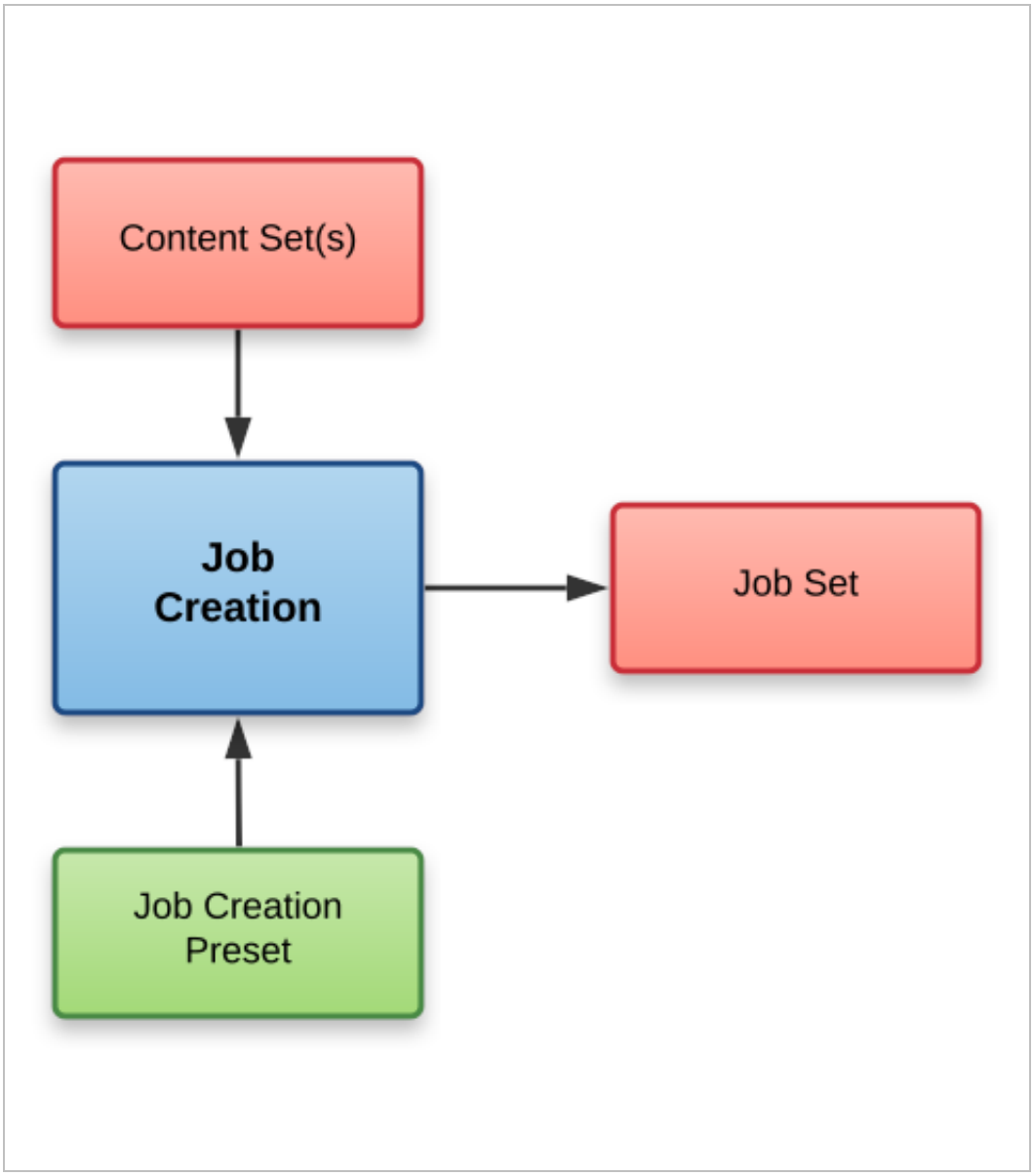
The following diagram illustrates the workflow for content creation of **Preview PDF**,by **JSON Data**:



Job Creation

The job creation process involves taking one or more content sets and applying a job creation preset for organizing, sorting and grouping them into a set of logical jobs (a job set). This includes the application of data filtering and finishing options.

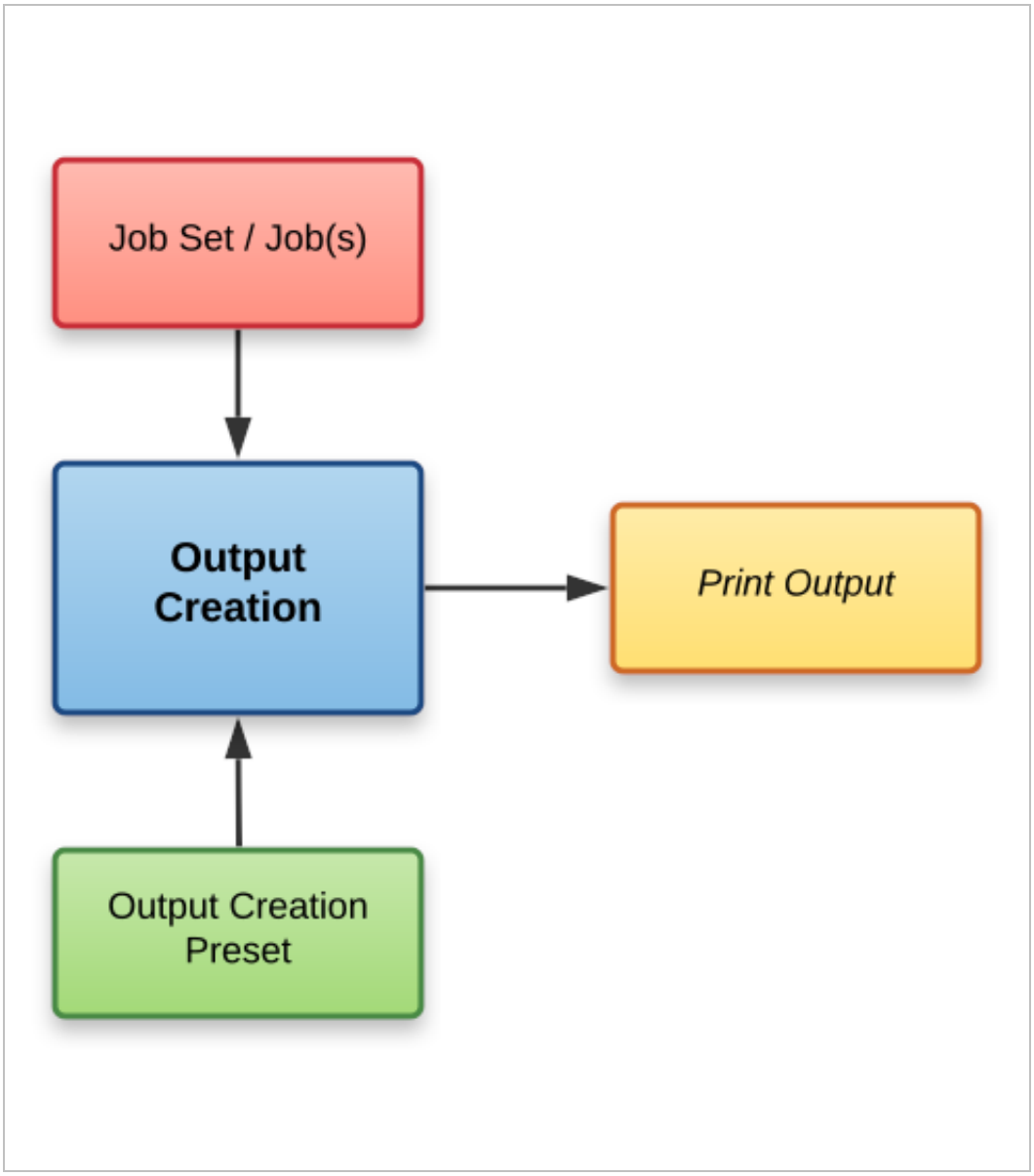
The following diagram illustrates the workflow for the job creation process:



Output Creation

The output creation process involves taking either a job set or one or more jobs (from a job set), applying an output creation preset, and producing the print output (Print context).

The following diagram illustrates the workflow for the output creation process:

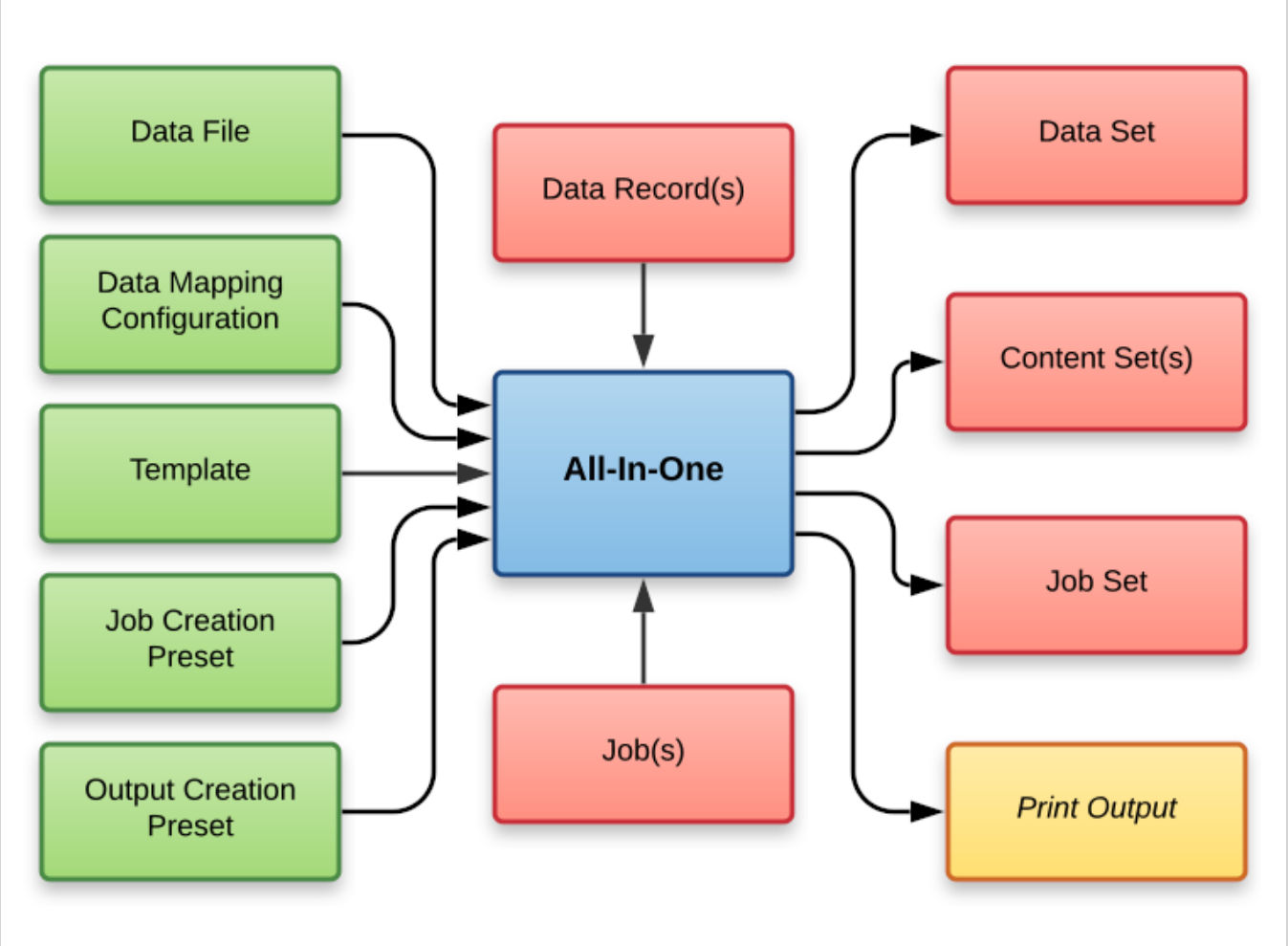


All-In-One

The All-In-One process embodies all four major workflow processes (data mapping, content creation, job creation and output creation) in a singular process. It can be configured to run one or more of the four processes, as long as the processes specified result in a logical sequence or workflow.

Depending on it's configuration, the All-In-One process can produce either a data set, content sets, a job set or print output (Print context).

The following diagram illustrates the potential inputs, outputs and workflows for the All-In-One process:



The following table lists the available processes, input combinations and expected outputs for the All-In-One process:

Processes	Input Combination	Expected Output
Data Mapping Only	Data File + Data Mapping Configuration	Data Set
Data Mapping + Content Creation	Data File + Data Mapping Configuration + Template	Content Set(s)

Processes	Input Combination	Expected Output
Content Creation Only	Data Records + Template	Content Set(s)
Data Mapping + Content Creation + Job Creation	Data File + Data Mapping Configuration + Template	Job Set
Data Mapping + Content Creation + Job Creation	Data File + Data Mapping Configuration + Template + Job Creation Preset	Job Set
Content Creation + Job Creation	Data Records + Template	Job Set
Content Creation + Job Creation	Data Records + Template + Job Creation Preset	Job Set
Data Mapping + Content Creation + Job Creation + Output Creation	Data File + Data Mapping Configuration + Template + Output Creation Pre-set	Print Output
Data Mapping + Content Creation + Job Creation + Output Creation	Data File + Data Mapping Configuration + Template + Job Creation Preset + Output Creation Preset	Print Output
Content Creation + Job Creation + Output Creation	Data Records + Template + Output Creation Preset	Print Output
Content Creation + Job Creation + Output Creation	Data Records + Template + Job Creation Preset + Output Creation Preset	Print Output
Output Creation Only	Jobs + Output Creation Preset	Print Output

Input Files

Input files are used as input to a specific workflow process. The following table lists the types of input files used in the PReS Connect workflow:

Name	Relevant Workflow Process	File Name Examples
Data File	Data Mapping (& Content Creation for PDF Preview)	• Promo-EN-10.csv • Promo-EN-10000.csv • PDFVT-Data.pdf
Data Mapping Configuration	Data Mapping (& Content Creation for PDF Preview)	• Promo-EN.OL-datamapper • Transact-EN.OL-datamapper
Template	Content Creation	• letter-ol.OL-template • invoice-ol-transpromo.OL-template
Job Creation Preset	Job Creation	• Promo-EN-JC-Config.OL-jobpreset
Output Creation Preset	Output Creation	• FX4112_Hold_Config.OL-outputpreset • Promo-EN-OC-Config.OL-outputpreset

Data Entities

There are many data entity types used by PReS Connect, but not all data entities can be accessed through the REST API. The main data entities to be aware of when working with the API are:

- Data Sets
- Data Records
- Content Sets
- Content Items
- Job Sets
- Jobs

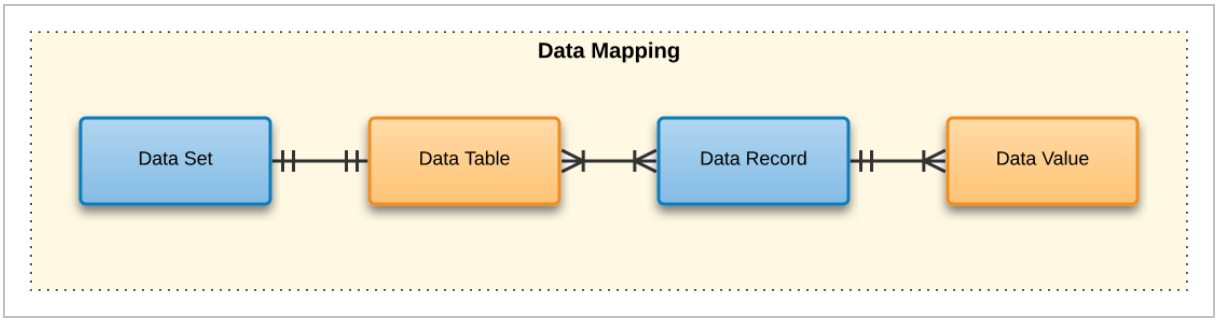
Data Set & Data Record Entities

The *data set* is the artefact produced by a data mapping operation. It holds the data that was mapped out of the input data file. A data mapping operation produces a single data set, which contains as many *data records* as there are documents.

Each data record contains a collection of *data values*. The data records in the data set form the master record, or document record, which typically contains document recipient information. The master record can also contain a collection of *data tables*, which form the detail records that hold data such as invoice line items.

Each data table contains a collection of data records, where each data record contains a collection of data values and a collection of data tables, and so on.

The following diagram illustrates the basic relationship between these entities in the context of the data mapping process:



The data set and data record entities (shown above in *blue*) are accessible via the Data Set Entity and Data Record Entity services.

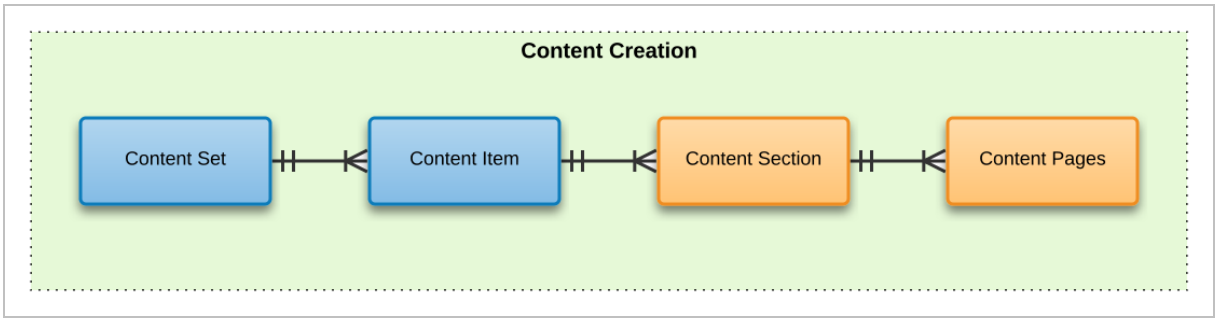
Content Set & Content Item Entities

The *content set* is the artefact produced by a content creation operation. It holds all the pages that were produced by the operation. A content creation operation produces one or more content sets, which

contain as many *content items* as there were data records given at the start of the operation.

Because the data records used may have different data set owners, a content set cannot be linked to a single data set, but rather content items are linked to data records. A content item is further divided into *content sections* and *content pages*.

The following diagram illustrates the basic relationship between these entities in the context of the content creation process:



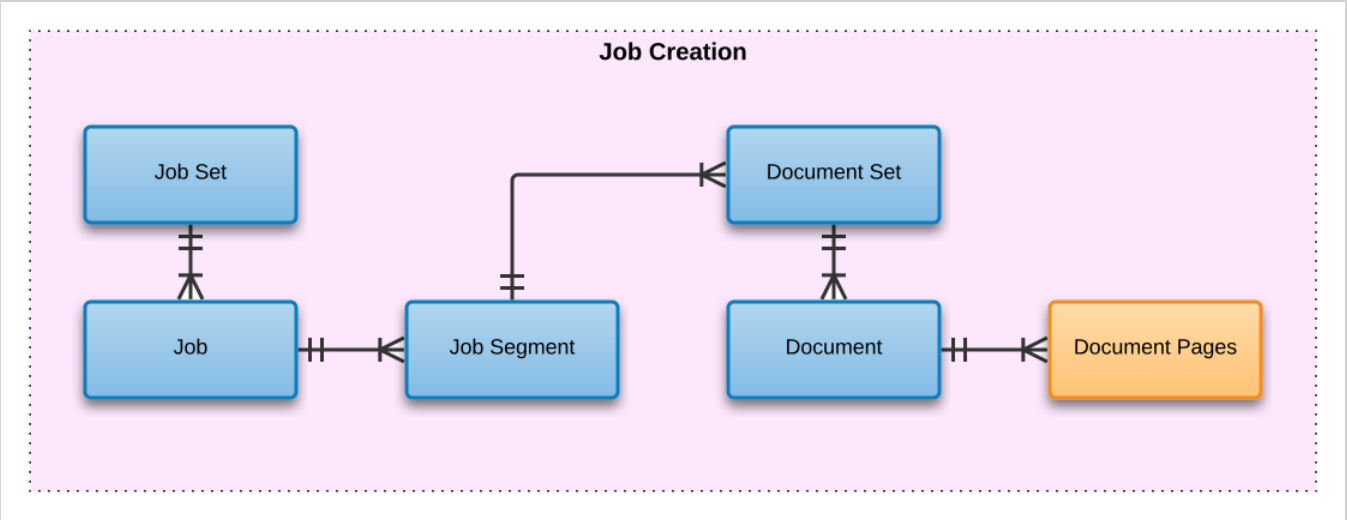
The content set and content item entities (shown above in *blue*) are accessible via the Content Set Entity and Content Item Entity services.

Job Set & Job Entities

The *job set* is the artefact produced by a job creation operation. It consists of a hierarchical structure that divides documents into various structures and it basically decides which documents are to be printed and in what order.

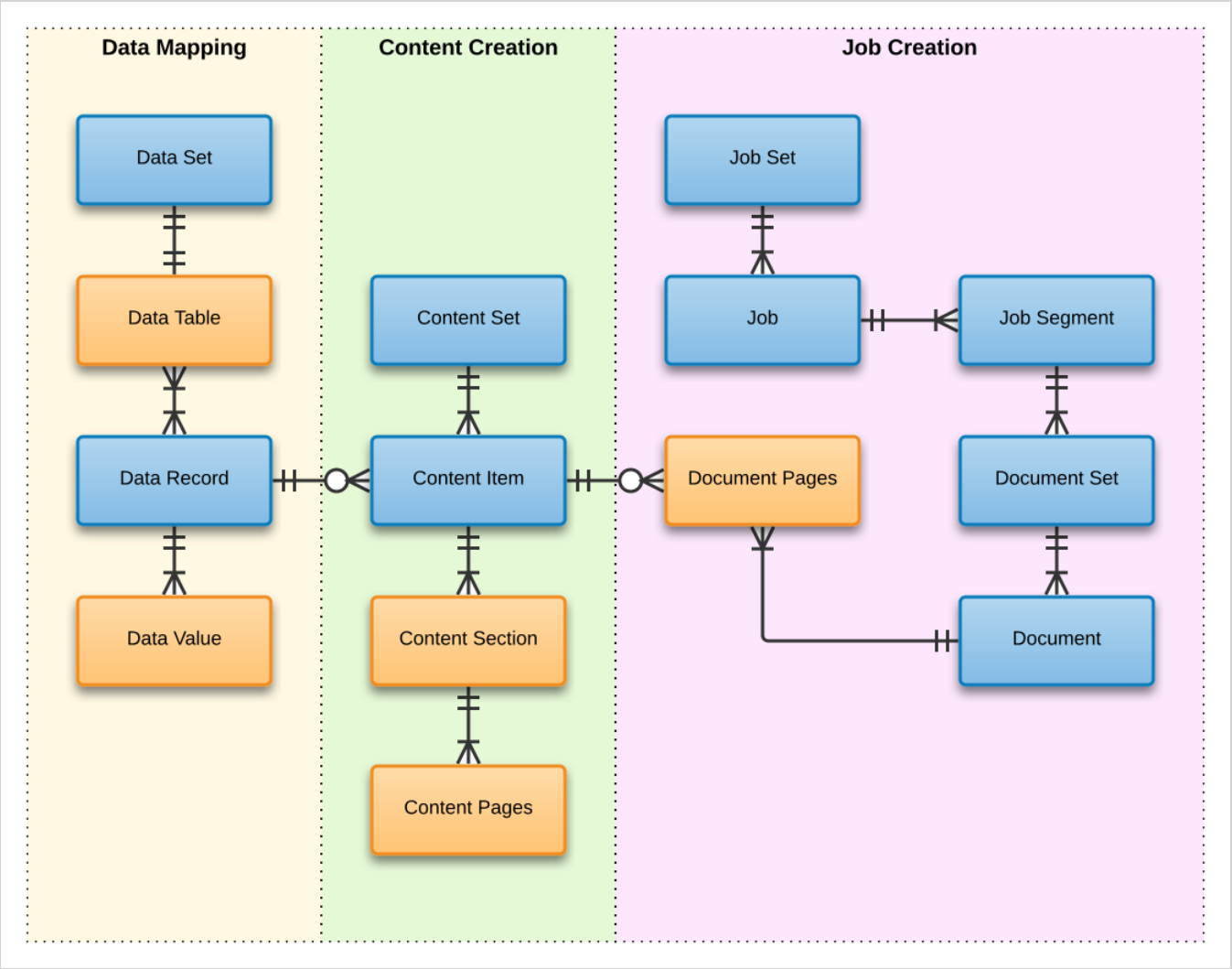
A job creation operation creates a single job set which contains a series of containers where every level contains one or more of the next level down: *jobs*, *job segments*, *document sets*, *documents* and *document pages*. The last level in the chain, the document pages, contains a single content item. Hence, at the job creation level, a document may consist of one or more content items.

The following diagram illustrates the basic relationship between these entities in the context of the job creation process:



The job set and job entities (shown above in *blue*) are accessible via the Job Set Entity and Job Entity services. The job segment, document set and document entities (also shown above in *blue*) are accessible via the Job Segment Entity, Document Set Entity and Document Entity services.

In summary, the following diagram illustrates the basic relationship between *all data entities* in the overall context of the primary workflow in PReS Connect:



Workflow Operations

Each individual process in the overall workflow can potentially be a long running operation.

Accordingly, there are two types of workflow operations possible in the PReS Connect REST API:

- **Asynchronous** – the operation is initiated, monitored, and the result returned using multiple requests (Default)
- **Synchronous** – the operation is initiated and the result returned using a single request

Asynchronous Operations

Asynchronous workflow operations require the submission of an initial HTTP request to initiate the operation. Then additional requests are required to monitor progress and retrieve the final result. All the required detail is included in the HTTP response headers of the initial request, including the URIs that should be used for further processing.

A successful request will return a response that will include the headers listed in the following table:

Header	Description
operationId	The unique id of the operation being processed
Link	Contains multiple link headers which provide details on which URI to use to retrieve further information on the operation: • Header with **rel="progress"** – The URL to use to check the progress of the operation • Header with **rel="result"** – The URL to use to retrieve the result of the operation • Header with **rel="cancel"** – The URL to use to cancel the operation

A request made to the **progress** URI during processing will return a progress percentage value of 0 to 100, and finally the value of 'done' once the operation has completed.

A request made to the **cancel** URI during processing will immediately cancel the operation.

A request made to the **result** URI after processing has completed will return the final result of the operation.

This is the default workflow operation type, and this approach is used across most workflow based services as demonstrated in the Working with the Workflow Services page of the Working Examples section.

Synchronous Operations

Synchronous workflow operations initiate the operation and retrieve the final result in a single request.

There are no additional operation related headers returned, and there is no option to either monitor progress or cancel a running operation.

This approach is only used by specific methods found in the All-In-One workflow service.

JSON Structures

The PReS Connect REST API uses various JSON structures to describe certain inputs and outputs to resource methods.

These structures can be broken down into the following categories:

- Common Structures – JSON structures that are commonly used throughout the REST API
- Specific Structures – JSON structures that are used by a specific resource method or service in the REST API

Common Structures

Common JSON structures used in the PReS Connect REST API include the following:

JSON Error

Describes an error response returned from the Connect server.

Structure

The structure consists of an object with the following name/value pair:

- `error` – the error response, consisting of an object with the following name/value pairs:
  - `status` – the response HTTP status code (*type* of `number`)
  - `message` – a short description of the error that occurred (*type* of `string`)
  - `parameter` – the ID of the resource that caused the error (*type* of `string`)

Example

The following is an example of this structure:

```
{
    "error": {
        "status": 404,
        "message": "The specified Job Set passed into this method refers to a
missing resource.",
        "parameter": "12345"
    }
}
```

JSON Identifier

Describes an identifier for a single data entity in PReS Connect.

Structure

The structure consists of an `object` with a single name/value pair:

- `identifier` – the data entity identifier (*type* of `number`)

Example

The following is an example of this structure:

```
{
    "identifier": 12345
}
```

JSON Identifier List

Describes a list of identifiers for multiple data entities in PReS Connect.

Structure

The structure consists of an `object` with a single name/value pair:

- `identifiers` – an `array` of data entity identifiers (*type* of `number`)

Example

The following is an example of this structure:

```
{
    "identifiers": [ 12345, 23456, 34567 ]
}
```

JSON Name/Value List

Describes a list of properties (each as a name/value pair) for a data entity of a specific ID.

Structure

The structure consists of an `object` with the following name/value pairs:

- `id` – the data entity identifier (*type* of `number`)
- `properties` – the data entity properties, consisting of an `array` of `objects` each with the following name/value pairs:
    - `name` – the name of the property (*type* of `string`)
    - `value` – the value of the property (*type* of `string`)

Example

The following is an example of this structure:

```
{
    "id": 12345,
    "properties": [
        {
            "name": "start",
            "value": "2015-01-01 00:00:00T-0500"
        },
        {
            "name": "end",
            "value": "2015-12-31 23:59:59T-0500"
        }
    ]
}
```

JSON Name/Value List (Properties Only)

Describes a list of properties (each as a name/value pair).

Structure

The structure consists of an `array` of `objects` each with the following name/value pairs:

- `name` – the name of the property (*type* of `string`)
- `value` – the value of the property (*type* of `string`)

Example

The following is an example of this structure:

```
[
    {
        "name": "start",
        "value": "2015-01-01 00:00:00T-0500"
    },
    {
        "name": "end",
        "value": "2015-12-31 23:59:59T-0500"
    }
]
```

JSON Name/Value Lists

Describes multiple lists of properties (as name/value pairs) for data entities of a specific ID.

Structure

The structure consists of an `array` of JSON Name/Value List structure `objects`.

Example

The following is an example of this structure:

```
[
    {
        "id": 12345,
        "properties": [
            {
                "name": "start",
                "value": "2015-01-01 00:00:00T-0500"
            },
            {
                "name": "end",
                "value": "2015-12-31 23:59:59T-0500"
            }
        ]
    },
    {
        "id": 23456,
        "properties": [
            {
                "name": "start",
                "value": "2015-01-01 00:00:00T-0500"
            },
            {
                "name": "end",
                "value": "2015-12-31 23:59:59T-0500"
            }
        ]
    }
]
```

Specific Structures

Specific JSON structures used in the PReS Connect REST API include the following:

- JSON Record Content List (Explicit Types)
- JSON Record Content List (Fields Only)
- JSON Record Content Lists
- JSON Record Content Lists (Explicit Types)
- JSON Record Content Lists (Fields Only)
- JSON Record Data List
- JSON Record Data List (with Email Parameters)
- JSON Record Data List (with Image Parameters)
- JSON Search Parameters

JSON All-In-One Configuration

Describes the configuration of an All-In-One operation as a series of name/value pairs representing the processes (data mapping, content creation, job creation and output creation) to be completed as part of the overall operation. The value in each pair contains the parameters for that specific process.

The structure is variable, allowing for configurations containing one or more specific processes (as name/value pairs), as long as the processes specified result in a logical sequence or workflow. Used specifically with the All-In-One service.

Structure

The structure consists of an `object` with the following name/value pairs:

- `datamining` – data mapping configuration parameters, consisting of an `object` with the following name/value pairs:
    - `identifier` – the managed file identifier (*type* of `number`) or named identifier (*type* of `string`) of the data file
    - `config` – the managed file identifier (*type* of `number`) or named identifier (*type* of `string`) of the data mapping configuration
    - `parameters` (*optional*) – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
        - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

- `contentcreation` – content creation configuration parameters, consisting of an `object` with the following name/value pairs:
    - `identifiers` – an `array` of data record entity identifiers (*type* of `number`) (optional for configurations containing data mapping parameters)
    - `config` – the managed file identifier (*type* of `number`) or named identifier (*type* of `string`) of the input template
    - `parameters` (*optional*) – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
        - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

- `jobcreation` – job creation configuration parameters, consisting of an `object` with the following name/value pairs:
    - `config` – the managed file identifier (*type* of `number`) or named identifier (*type* of `string`) of the job creation preset (optional)
    - `parameters` (*optional*) – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
        - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

- `outputcreation` – output creation configuration parameters, consisting of an `object` with the following name/value pairs:
    - `identifiers` – an `array` of job entity identifiers (*type* of `number`) (optional for configurations containing content creation parameters)
    - `config` – the managed file identifier (*type* of `number`) or named identifier (*type* of `string`) of the output creation preset
    - `createOnly` – flag to specify if output is to be only created in the server and not sent to it's final destination (*type* of `boolean`)

Specific to the use of all processes, an additional name/value pair can be added to restrict the print output to a set of specific records in the input data:

- `printRange` – print range configuration parameters, consisting of an `object` with a single name/value pair:
    - `printRange` – the range of records in the data file to output (*type* of `string`)

Specific to any configuration using the data mapping process, but with the **omission** of the `config` job creation configuration parameter (if applicable), an additional data mapping entry can be added to the `datamining` object:

- `persistDataset` – parameter to specify if data record entities are to be created/persisted in the server during the data mapping process (*type* of `boolean`)

Example

The following are examples of this structure:

```
{
        "datamining":
       {
```

```
                "identifier": "Promo-EN-1000.csv",
                "config": "Promo-EN.OL-datamapper",
                "parameters": {
                        "Gender": "Female"
                }
        },
        "contentcreation":
        {
                "config": "letter-ol.OL-template",
                "parameters": {
                        "InvoiceDueDate": "2020-03-10",
                }
        },
        "jobcreation":
        {
                "config": "4567",
                "parameters": {
                        "TrackingId": "20211"
                }
        },
        "outputcreation":
        {
                "config": "5678",
                "createOnly": true
        },
        "printRange":
        {
                "printRange": "1-3, 6, 10"
        }
}
```

```
{
    "contentcreation":
    {
        "identifiers": [
            34567,
            34568
        ],
        "config": "letter-ol.OL-template"
```

```
    },
    "jobcreation": {},
    "outputcreation":
    {
        "config": "5678",
        "createOnly": false
    }
}

{
    "datamining":
    {
        "identifier": "12345",
        "config": "23456"
    }
}

{
    "datamining":
    {
        "identifier": "Promo-EN-1000.csv",
        "config": "Promo-EN.OL-datamapper",
        "persistDataset": false
    },
    "contentcreation":
    {
        "config": "letter-ol.OL-template"
    },
    "jobcreation": {},
    "outputcreation":
    {
        "config": "5678",
        "createOnly": false
    }
}

{
    "contentcreation":
    {
        "identifiers": [
```

```
            34567,
            34568
        ],
        "config": "letter-ol.OL-template"
    },
    "jobcreation": {
        "config": "4567",
        "parameters": {
            "FirstName": "Benjamin",
            "InvoiceDueAmount": 123.45,
            "InvoiceDueDate": "2020-03-10",
            "InvoiceOverdue": true
        }
    },
    "outputcreation":
    {
        "config": "5678",
        "createOnly": false
    }
}
```

JSON Bitmap Parameters

Describes a list of parameters used specifically in the rasterization of a PDF, PS, AFP or PCL file, defining which pages of the source file have to be converted to a bitmap and the properties of the resulting bitmap(s).

Structure

The structure consists of an `object` with the following optional name/value pairs:

- `type` – the image type/format to be used in the creation of the image (*value* of either `jpg`, `jpeg` or `png`)
  (*type* of `string` – *default value* of `jpg`)
- `pages` – the page range to be output; this counts over all pages even if one of the sections is configured to restart page numbers
  (*type* of `string` – *default value* is determined by the value of the `archive` parameter. If the `archive` parameter is specified to `false`, then the default value will be `1`. If the archive parameter is either omitted or specified to a value of `true`, then the default value will be * (all pages))

- `archive` – whether to return the result as a ZIP file/archive; `true` means return the result as ZIP file, `false` as a JPEG or PNG file
  (*type* of `boolean` – *default value* is automatic: `false` if the output consists of a single file, `true` if the output consists of multiple files)
- `dpi` – the target image resolution in *dots per inch* (DPI)
  (*type* of `number` – *default value* of `96`)
- `scale` – the scale of the target image
  (*type* of `float` – *default value* of `1.0`)

Specific to parameters with a `type` parameter specified to a *value* of `jpg`, the following optional name/-value pair can be specified:

- `quality` – the image quality of the preview (*value* ranging from `0-100`)
  (*type* of `number` – *Default value* of `100`)

Example

The following is an example of this structure:

```
{
    "type": "png",
    "dpi": 150,
    "archive": true,
    "pages": "1,3,6-11"
}

{
    "type": "jpg",
    "quality": 90
}
```

JSON Content Item Identifier List

Describes a list of content item/data record entity identifier pairs (as name/value pairs) for a specific content set or job entity.

Structure

The structure consists of an `object` with the following name/value pairs:

- `identifiers` – the data entity identifier pairs, consisting of an `array` of `objects` each with the following name/value pairs:
  - `item` – the content item entity identifier (*type* of `number`)
  - `record` – the data record entity identifier (*type* of `number`)

Example

The following is an example of this structure:

```
{
    "identifiers": [
        {
            "item": 12345,
            "record": 54321
        },
        {
            "item": 23456,
            "record":  65432
        },
        {
            "item": 34567,
            "record": 76543
        }
    ]
}
```

JSON Data Mapping Validation Result

Describes the result of a request to validate a data mapping operation, including a list of any errors that occurred (used specifically with the Data Mapping service).

Structure

The structure consists of an `object` with the following name/value pairs:

- `result` – the overall result of the data mapping operation (*value* of either `ERROR` or `OK`) (*type* of `string`)
- `recordcount` – the number of data records in the data file (*type* of `number`)
- `errors` – a list of errors that occurred during the mapping process, consisting of an `array` of `objects` each with the following name/value pairs:
    - `record` – the number of the erroneous record in the data file (*type* of `number`)
    - `reason` – the mapping error/reason for this particular record (*type* of `string`)

Example

The following is an example of this structure:

```
{
    "result": "ERROR",
    "recordcount": 105,
    "errors": [
        {
            "record": 20,
            "reason": "Document: 20 Unparseable date: \"\""
        },
        {
            "record": 45,
            "reason": "Document: 45 Unparseable date: \"\""
        },
        {
            "record": 97,
            "reason": "Document: 97 Unparseable date: \"\""
        }
    ]
}
```

JSON Data Record Identifier

Describes a single data record entity identifier for a specific content item entity.

Structure

The structure consists of an `object` with a single name/value pair:

- `record` – the data record entity identifier (*type* of `number`)

Example

The following is an example of this structure:

```
{
    "record": 12345
}
```

JSON Data Record Identifier List (with Parameters)

Describes a list of identifiers for multiple data entities (specifically data record entities), along with additional parameters.

It is used specifically with the Data Record Entity service as input to the Get Multiple Data Record Values (JSON) resource method.

The *value* of the `outputFormat` parameter determines if the result returned is a JSON Record Content Lists or JSON Record Content Lists (Explicit Types) structure.

Structure

The structure consists of an `object` with the following name/value pairs:

- `recordids` – an `array` of data record entity identifiers (*type* of `number`)
- `recursive` – parameter to specify if all data tables within each data record should be recursed and the values of any nested data records retrieved also (*type* of `boolean`)
- `explicitTypes` (optional) – ***deprecated; use*** `outputFormat` ***instead***. Parameter to specify if both data record values and data types are to be retrieved (*type* of `boolean`)
- `outputFormat` (optional) - Parameter to specify the output format of the data (*value* of either `default`, `explicitTypes` or `simplified`) (*type* of `string`). Default: `default`.
- `optimized` (optional) – parameter to specify whether the method can retrieve items in an optimized way.
  If `true`, the resulting JSON is streamed back to the client as soon as the first record is available to return. Duplicate record IDs from the request are ignored: for any duplicated record ID in a list of requested IDs, a single entry is returned. The resulting JSON does not respect the order of requested record IDs. The resulting JSON has ASC order based on the data record IDs.
  Default: `false`.
- `batchsize` (*optional*) - parameter to specify a custom batch size. For smaller records the batch size can be over 1000; for bigger records it has to be less. Default: 1000.

Example

The following is an example of this structure:

```
{
    "recordids": [ 12345, 23456, 34567 ],
    "recursive": true,
    "outputFormat": "simplified",
    "optimized": true,
```

```
    "batchsize": 1200
}
```

JSON Email Output List

Describes a list of email message output, a content set ID for that output, and any errors that may have occurred specific to a content creation for email operation.

Structure

The structure consists of an `object` with the following name/value pairs:

Specific to email output directly to a SMTP mail server, the following name/value pair will be specified:

- `messages` – a list of the email messages created, consisting of an array of objects each with the following name/value pairs:
    - `subject` – the subject of the email message (*type* of `string`)
    - `to` – the email addresses of the recipients in the email message (*type* of `string`)
    - `body` – the name of the HTML email body file in the email message output (*type* of `string`)
    - `folder` – the managed file ID for the output directory in the file store containing the email message output (*type* of `number`)
    - `attachments` – a list of the file attachments to the email message, consisting of an array of objects each with the following name/value pairs:
        - `name` – the name of the file attachment in the email message output (*type* of `string`)
        - `disposition` – the disposition of the attachment to the email message (*type* of `string`)

    Specific to inclusion of meta tags in the template's scripts, the following optional name/value pairs can be specified:

    - `from` – the email address of the sender in the email message (*type* of `string`)
    - `cc` – the email addresses of the carbon copy (CC) recipients in the email message (*type* of `string`)
    - `bcc` – the email addresses of the blind carbon copy (BCC) recipients in the email message (*type* of `string`)
    - `replyTo` – the email addresses to reply to in the email message (*type* of `string`)

- `headers` – a list of any custom email headers in the email message, consisting of an array of objects each with the following name/value pairs:
    - `<name>` – the name (*name*) and value of the email header (*type* of `string`)

  Specific to email message output where an EML (E-Mail Message) file is to be created, the following name/value pair will be specified:

  - `eml` – the name of the EML (E-Mail Message) file in the email message output (*type* of `string`)

  Specific to templates with an email context section configured to append a plain-text copy of the HTML, the following name/value pair will be specified:

  - `text` – the name of the TXT email file in the email message output (*type* of `string`)
- `errors` – a list of any errors that occurred during content creation, consisting of an array of objects each with the following name/value pairs:
  - `record` –the data record index associated with the error (*type* of `number`)
  - `message` –the description of the error that occurred (*type* of `string`)
- `contentSet` – the content set entity identifier for the email output (*type* of `number`)
- `successCount` - the number of messages that were successfully processed or sent.

Examples

The following is an example of this structure:

```
{
    "messages": [
        {
            "attachments": [
                {
                    "name": "att97529c81-f882-4c21-8b39-a404f728d43e.png",
                    "disposition": "inline"
                },
                {
                    "name": "att2fa0f059-ec10-4054-b859-36ad1f2fdf59.png",
                    "disposition": "inline"
                },
                {
                    "name": "att86c7b3c2-3483-4926-b893-608a42231d70.jpg",
```

```
            "disposition": "inline"
        },
        {
            "name": "attf086d7d3-d33f-478a-b1a6-771deeb39b5b.png",
            "disposition": "inline"
        },
        {
            "name": "attb0db370d-db4d-4bc3-93be-82438603c253.png",
            "disposition": "inline"
        },
        {
            "name": "att4c7f31d9-d512-42b6-b0a9-6e120c07701f.png",
            "disposition": "inline"
        },
        {
            "name": "att80b0c7c4-6aaf-45b8-b559-b5f0e3c4e378.png",
            "disposition": "inline"
        },
        {
            "name": "att7d18e853-8013-46c3-819a-2efdad883a10.png",
            "disposition": "inline"
        },
        {
            "name": "John, gain complete control!.html",
            "disposition": "attachment"
        }
    ],
    "subject": "John, gain complete control!",
    "to": "k.snell@drupa.ol.com.com",
    "folder": 12345,
    "eml": "63cb15da-42be-4dd8-88e3-367e0f3c9fec.eml",
    "body": "21545016-ef51-4e36-8932-73b0897c4672.html"
},
{
    "attachments": [
        {
            "name": "attb374b927-5b0a-48d6-9fd0-678a0c6b1faf.png",
            "disposition": "inline"
```

```
        },
        {
            "name": "att264e569e-0f13-4422-a1a5-41c9cda0797e.png",
            "disposition": "inline"
        },
        {
            "name": "att35d7d743-886d-405d-b5d4-ce73d43b9de3.jpg",
            "disposition": "inline"
        },
        {
            "name": "att9eb5d506-f9ac-4df0-9b72-e164f4dd1237.png",
            "disposition": "inline"
        },
        {
            "name": "att357f4a10-047c-45a7-b292-0862bd50979d.png",
            "disposition": "inline"
        },
        {
            "name": "att335b33e2-4ca1-4d2f-a7ea-06ca96c1544b.png",
            "disposition": "inline"
        },
        {
            "name": "att48c026ef-0bdc-4709-ac71-eea25ec5f137.png",
            "disposition": "inline"
        },
        {
            "name": "att063ad47c-dc26-4105-b013-9dacb4e47718.png",
            "disposition": "inline"
        },
        {
            "name": "Kristopher, gain complete control!.html",
            "disposition": "attachment"
        }
    ],
    "subject": "Kristopher, gain complete control!",
    "to": "k.snell@drupa.ol.com.com",
    "folder": 23456,
    "eml": "9999069e-28dd-4182-86a9-81cb0a228261.eml",
```

```
            "body": "124ffa76-8794-4b1e-afb7-2f886738c05f.html"
        }
    ],
    "errors": [],
    "contentSet": 34567,
    "successCount": 2
}
```

JSON Email Parameters (with Runtime Parameters)

Describes runtime parameters and parameters used specifically in an content creation operation for email.

This structure is variable, allowing specification for email output to either the File Store or directly to a SMTP mail server, with a number of additional parameters.

Structure

A subset of the runtime parameters defined in the template can be passed in the following object:

- `parameters` – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

For either email output directly to a SMTP mail server or email output to the file store, the following optional name/value pairs can be specified:

- `attachPdfPage` – parameter to specify if a PDF file of the Print context should also be created and attached to the email output (*type* of `boolean`)
- `attachWebPage` – parameter to specify if HTML files of the enabled sections (a single section by default) in the Web context should also be created and attached to the email output (*type* of `boolean`)
- `addPlainText` - parameter to specify if a plain text copy should be added to the email (*type* of `boolean`). This parameter overrides any setting in the template.

Specific to email output directly to a SMTP mail server, an additional name/value pair is required:

- `host` – the network address or name of the SMTP mail server through which emails will be sent. If required, a server port value can also be specified (*type* of `string`)

Specific to email output directly to a SMTP mail server, an optional name/value pair can be specified:

- `useAuth` – parameter to specify if authentication is to be used with the mail server (*type* of `boolean`)

Specific to email output directly to a SMTP mail server *with* the useAuth parameter specified to a value of `true`, the following optional name/value pairs can be specified:

- `user` – the user name to authenticate with (*type* of `string`)
- `password` – the password to authenticate with (*type* of `string`)
- `useStartTLS` – parameter to specify if Transport Layer Security (TLS) is to be opportunistically used when sending emails (*type* of `boolean`)

Specific to email output to the File Store, an optional name/value pair can be specified:

- `eml` – parameter to specify if an EML (E-Mail Message) file of the email for each record should be created in the email output (*type* of `boolean`)

Specific to either email output directly to a SMTP mail server or email output to the file store with the eml parameter specified to a value of true, an additional name/value pair is required:

- `sender` – the email address to be shown as the sender in the email output (*type* of `string`)

Specific to either email output directly to a SMTP mail server or email output to the file store with the sender parameter specified, the following optional name/value pairs can be specified:

- `senderName` – the name to be shown as the sender in the email output (*type* of `string`)
- `useSender` – parameter to specify if the sender address will be used as the receiver address for all emails in the output (*type* of `boolean`)

Examples

The following are examples of this structure:

```
{
    "identifiers": [
        12345,
        23456
    ],
    "attachPdfPage": true,
    "attachWebPage": true,
    "sender": "john.smith@company.com",
    "useSender": true,
    "host": "mail.company.com",
    "useAuth": true,
```

```
    "user": "johns",
    "password": "password5",
}

{
    "identifiers": [
        12345,
        23456,
        34567
    ],
    "attachWebPage": true,
    "sender": "jane.smith@company.com",
    "senderName": "Jane Smith",
    "eml": true
}

{
    "identifiers": [
        12345,
        23456,
        34567,
        45678
    ],
    "attachPdfPage": false,
    "attachWebPage": true
}
```

JSON HTML Parameters

Describes a list of parameters used specifically in the creation of web content.

Structure

The structure consists of an `object` with the following name/value pairs:

- `section` – the section within the Web context of the template to use (*type* of `string`)
- `inline` – the inline mode to be used in the creation of content (*type* of `string`)
  (*value* of either:
  - `NONE` - no inlining
  - `CSS` - converts style rules to inline styles on elements
  - `ALL` - inline all resources
  - `LOCAL` - inline local resources; remote resources remain external)
- `cssSelector` – a CSS selector for the creation of only a specific HTML element within the template (*type* of `string`)

Example

The following is an example of this structure:

```
{
    "section": "Section 1",
    "inline": "ALL"
}

{
    "section": "Section 2",
    "cssSelector": "#salutation"
}
```

JSON HTML Parameters (with Runtime Parameters)

Describes a list of parameters used specifically in the creation of web content.

Structure

The structure consists of an `object` with the following name/value pairs:

- `section` – the section within the Web context of the template to use (*type* of `string`)
- `inline` – the inline mode to be used in the creation of content (*type* of `string`)
  (*value* of either:

    - `NONE` - no inlining
    - `CSS` - converts style rules to inline styles on elements
    - `ALL` - inline all resources
    - `LOCAL` - inline local resources; remote resources remain external)
- `cssSelector` – a CSS selector for the creation of only a specific HTML element within the template (*type* of `string`)
- `parameters` – a set of runtime parameter names (defined in the template) and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

Example

The following are examples of this structure:

```
{
    "section": "Section 1",
    "inline": "ALL"
}

{
    "section": "Section 2",
    "cssSelector": "#salutation"
}
```

JSON Identifier (for Content Creation)

Describes an identifier for a data mapping configuration, along with parameters used in a content creation operation and runtime parameters used in the data mapping configuration.

Structure

The structure consists of an `object` with the following name/value pairs:

- `identifier` – File Store ID of the data mapping configuration (*type* of `number`)
- `defaults` (*optional*) – default properties for Content Creation, consisting of an object with one or more name/value pairs:

    - `duplex` – whether the page sheet is duplex (*type* of `boolean`). Default: `false`.
    - `tumble` – whether to duplex pages as in a calendar (*type* of `boolean`). This requires `duplex` to be set to `true`. Default: `false`.
- `parameters` (*optional*) – a set of runtime parameter names and their corresponding values, used in the data mapping configuration. The set consists of an object with one or more name/-value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)
- `range` (*optional*) - a range of records in the data file (type of `string`).

Example

The following is an example of this structure:

```
{
        "identifier": 12345,
        "defaults": {
                "duplex": "true",
                "tumble": "true"
        },
        "range": {"1, 5-10"}
}
```

JSON Identifier (Managed File)

Describes an identifier or named identifier for a single managed file in PReS Connect.

Structure

The structure consists of an `object` with a single name/value pair:

- `identifier` – the managed file identifier (*type* of `number`) or named identifier (*type* of `string`)
- `parameters` (*optional*) – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

Example

The following are examples of this structure:

```
{
    "identifier": 12345
}

{
    "identifier": "Promo-EN-1000.csv"
}
```

JSON Identifier (with Output Parameters)

Describes an identifier for a single job set entity, along with additional parameters used specifically in an output creation operation.

Structure

The structure consists of an `object` with the following name/value pairs:

- `identifier` – the job set entity identifier (*type* of `number`)
- `createOnly` – parameter to specify if output is to be only created in the server and not sent to it's final destination (*type* of `boolean`)

Example

The following is an example of this structure:

```
{
    "identifier": 12345,
    "createOnly": true
}
```

JSON Identifier (with Runtime Parameters)

Describes a list of identifiers for multiple content set entities, along with additional runtime parameters used specifically in a job creation operation.

Structure

The structure consists of an `object` with the following name/value pairs:

- `identifier` – an `array`) of content set entity identifiers.
  (*type* of `number`)
- `parameters` – a set of the runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
  - `<name>` – the name (name) and the value of the runtime parameter
    (*type* of either `string`, `number`, or `boolean`)

Example

The following is an example of this structure:

```
{
    "identifiers": [ 12345, 23456, 34567 ],
    "parameters": {
```

```
            "FirstName": "Benjamin",
            "InvoiceDueAmount": 123.45,
            "InvoiceDueDate": "2020-03-10",
            "InvoiceOverdue": true
        }
    }
```

JSON Identifier List (with Email Parameters)

Describes a list of identifiers for multiple data entities (specifically data record entities), along with additional parameters used specifically in an content creation operation for email.

This structure is variable, allowing specification for email output to either the File Store or directly to a SMTP mail server, with a number of additional parameters.

Structure

The structure initially consists of an object with the following name/value pair:

- `identifiers` – an array of data record entity identifiers (*type* of `number`)

In addition, a subset of the runtime parameters defined in the template can be passed in the following object:

- `parameters` – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

For either email output directly to a SMTP mail server or email output to the file store, the following optional name/value pairs can be specified:

- `attachPdfPage` – parameter to specify if a PDF file of the Print context should also be created and attached to the email output (*type* of `boolean`)
- `attachWebPage` – parameter to specify if HTML files of the enabled sections (a single section by default) in the Web context should also be created and attached to the email output (*type* of `boolean`)
- `addPlainText` - parameter to specify if a plain text copy should be added to the email (*type* of `boolean`). This parameter overrides any setting in the template.

Specific to email output directly to a SMTP mail server, an additional name/value pair is required:

- `host` – the network address or name of the SMTP mail server through which emails will be sent. If required, a server port value can also be specified (*type* of `string`)

Specific to email output directly to a SMTP mail server, an optional name/value pair can be specified:

- `useAuth` – parameter to specify if authentication is to be used with the mail server (*type* of `boolean`)

Specific to email output directly to a SMTP mail server *with* the useAuth parameter specified to a value of `true`, the following optional name/value pairs can be specified:

- `user` – the user name to authenticate with (*type* of `string`)
- `password` – the password to authenticate with (*type* of `string`)
- `useStartTLS` – parameter to specify if Transport Layer Security (TLS) is to be opportunistically used when sending emails (*type* of `boolean`)

Specific to email output to the File Store, an optional name/value pair can be specified:

- `eml` – parameter to specify if an EML (E-Mail Message) file of the email for each record should be created in the email output (*type* of `boolean`)

Specific to either email output directly to a SMTP mail server or email output to the file store with the eml parameter specified to a value of true, an additional name/value pair is required:

- `sender` – the email address to be shown as the sender in the email output (*type* of `string`)

Specific to either email output directly to a SMTP mail server or email output to the file store with the sender parameter specified, the following optional name/value pairs can be specified:

- `senderName` – the name to be shown as the sender in the email output (*type* of `string`)
- `useSender` – parameter to specify if the sender address will be used as the receiver address for all emails in the output (*type* of `boolean`)

Examples

The following are examples of this structure:

```
{
    "identifiers": [
        12345,
        23456
    ],
    "attachPdfPage": true,
    "attachWebPage": true,
    "sender": "john.smith@company.com",
    "useSender": true,
    "host": "mail.company.com",
    "useAuth": true,
    "user": "johns",
    "password": "password5",
}
```

```
{
    "identifiers": [
        12345,
        23456,
        34567
    ],
    "attachWebPage": true,
    "sender": "jane.smith@company.com",
    "senderName": "Jane Smith",
    "eml": true
}
```

```
{
    "identifiers": [
        12345,
        23456,
        34567,
        45678
    ],
    "attachPdfPage": false,
    "attachWebPage": true
}
```

JSON Identifier List (with Output Parameters)

Describes a list of identifiers for multiple job entities, along with additional parameters used specifically in an output creation operation.

Structure

The structure consists of an `object` with the following name/value pairs:

- `identifiers` – an `array` of job entity identifiers (*type* of `number`)
- `createOnly` – parameter to specify if output is to be only created in the server and not sent to it's final destination (*type* of `boolean`)

Example

The following is an example of this structure:

```
{
    "identifiers": [ 12345, 23456, 34567 ],
    "createOnly": true
}
```

JSON Identifier List (with Runtime Parameters)

Describes a list of identifiers for multiple data entities in PReS Connect.

Structure

The structure initially consists of an `object` with a single name/value pair:

- `identifiers` – an `array` of data entity identifiers (*type* of `number`)

In addition, a subset of the runtime parameters defined in the template can be passed in the following object:

- `parameters` – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

Example

The following is an example of this structure:

```
{
    "identifiers": [ 12345, 23456, 34567 ]
}
```

JSON Identifier Lists (with Sort Key)

Describes a set of search results as a list of one or more sub lists, each containing a list of data entity identifiers along with a sorting key value for each entry.

Used specifically with the Entity service as the output from the Find Data Entity resource method, this structure groups the data entity identifiers returned into sortable sub lists of entries.

The order of the entries (including the sort key produced), and the number of sub lists returned depends on the sorting and grouping rules specified in the JSON Search Parameters structure previously submitted as input to the Find Data Entity resource method.

Structure

The structure consists of an `array` of `object` `arrays`, with each `object` containing the following name/value pairs:

- `identifier` – the data entity identifier (*type* of `number`)
- `sortkey` – the data entity sort key (*type* of `string`)

Example

The following is an example of this structure:

```
[
    [
        {
            "identifier": 1604,
            "sortkey": "NB|Vilma"
        },
        {
            "identifier": 1282,
            "sortkey": "NF|Lenard"
        },
        {
            "identifier": 1443,
            "sortkey": "NF|Lenard"
        },
        {
            "identifier": 1000,
            "sortkey": "SK|Cathleen"
        },
        {
```

```
            "identifier": 1121,
            "sortkey": "SK|Rachel"
        }
    ]
]
```

JSON Image Parameters

Describes a list of parameters used specifically in the creation of a preview image of content for print, email or web.

This structure is used specifically by the Content Creation service when creating preview images.

Structure

The structure consists of an `object` with the following optional name/value pairs:

- `context` – the context to be used in the creation of the preview (*value* of either `print`, `email` or `web`)
  (*type* of `string` – *Default value* is determined by the first context in the template)
- `section` – the section to be used within the context specified (as either the `context` parameter, or else the default context of the template)
  (*type* of `string` – *Default value* is determined by the context specified. For the *Print* context this will be all enabled sections. For the *Email* and *Web* contexts this will be the default section)
- `type` – the image type/format to be used in the creation of the preview (*value* of either `jpg`, `jpeg` or `png`)
  (*type* of `string` – *Default value* of `jpg`)
- `dpi` – the target image resolution of the preview in *dots per inch* (DPI)
  (*type* of `number` – *Default value* of `96`)
- `archive` – whether to return the resulting preview as a ZIP file/archive
  (*type* of `boolean` – *Default value* is determined automatically by the number of image files in the preview output)

Specific to parameters with a `type` parameter specified to a *value* of `jpg`, the following optional name/-value pair can be specified:

- `quality` – the image quality of the preview (*value* ranging from `0-100`)
  (*type* of `number` – *Default value* of `100`)

Specific to parameters with a `context` parameter specified to a *value* of `print`, the following optional name/value pair can be specified:

- `bleed` – whether to include the bleed area in the preview
  (*type* of `boolean` – *Default value* of `false`)
- `pages` – the page range to be output in the preview
  (*type* of `string` – *Default value* is determined by the value of the `archive` parameter. If the `archive` parameter is specified to `false`, then the default value will be `1`. If the archive parameter is either omitted or specified to a value of `true`, then the default value will be * (all pages))

Specific to parameters with a `context` parameter specified to a *value* of either `email`, or `web` the following optional name/value pair can be specified:

- `viewPortWidth` – the image width of the preview in *pixels*
  (*type* of `number` – *Default value* of `1024`)

Example

The following is an example of this structure:

```
{
    "context": "print",
    "type": "png",
    "dpi": 150,
    "archive": true,
    "bleed": true,
    "pages": "1-2"
}

{
    "context": "web",
    "section": "Section 1",
    "type": "jpeg",
    "viewPortWidth": 1024
}

{
    "context": "email",
    "section": "Section 2",
    "type": "jpg",
    "quality": 90
}
```

JSON Job Set Structure

Describes a job set entity structure including the arrangement of job, job segment, document set, document and content item entities (including the specification of content item identifiers). Used specifically in a job creation operation.

Structure

The structure consists of an `object` with the following name/value pairs:

- `jobs` – the job entities within the job set, consisting of an `array` of `objects` each with the following name/value pairs:
  - `segments` – the job segment entities within a job, consisting of an `array` of `objects` each with the following name/value pairs:
    - `documentsets` – the document set entities within a job segment, consisting of an `array` of `objects` each with the following name/value pairs:
      - `documents` – the document entities within a document set, consisting of an `array` of `objects` each with the following name/value pairs:
        - `documentpages` – the document pages within a document, consisting of an `array` of `objects` each with a single name/value pair:
          - `contentitem` – the identifier of the content item entity within a document page (*type* of `number`)

Example

The following is an example of this structure:

```
{
    "jobs": [
        {
            "segments": [
                {
                    "documentsets": [
                        {
                            "documents": [
                                {
                                    "documentpages": [
                                        {
                                            "contentitem": 1234
                                        },
```

```
                                    {
                                        "contentitem": 2345
                                    }
                                ]
                            },
                            {
                                "documentpages": [
                                    {
                                        "contentitem": 3456
                                    }
                                ]
                            }
                        ]
                    }
                ]
            }
        ]
    },
    {
        "segments": [
            {
                "documentsets": [
                    {
                        "documents": [
                            {
                                "documentpages": [
                                    {
                                        "contentitem": 4567
                                    }
                                ]
                            }
                        ]
                    }
                ]
            }
        ]
    }
]
```

```
}
```

JSON New Record List

Describes a list of new data records (and their data field values (as name/value pairs)) to be added as data record entities to either an existing data set or data record entity of a specific ID.

> **Note:** Date/time values conforming to ISO 8601 are stored in the OL Connect database as Unix timestamps, i.e. the total number of seconds from the starting time of UTC (Universal Time, Coordinated) to the current time (with zero time zone offset). For instance: 1675950179.

Structure

The structure consists of an `object` with the following name/value pairs:

- `records` – a list of the new data records to be added, consisting of an `array` of `objects` each with the following name/value pairs:
    - `fields` – a list of data fields for the data record, consisting of an `array` of `objects` each with the following name/value pairs:
        - `name` – the name of the data field (*type* of `string`)
        - `value` – the value of the data field (*type* of `string`)

Specific to the adding of *data records* to the `record` data table of an existing *data set* entity, an additional name/value pair is included:

- `datasetid` – the data set entity identifier of parent entity (*type* of `number`)

Specific to the adding of *nested data records* to a data table of an existing *data record* entity, two additional name/value pairs are included:

- `recordid` – the data record entity identifier of parent entity (*type* of `number`)
- `table` – the data record entity data table name (*type* of `string`)

Example

The following are examples of this structure:

```
{
    "datasetid": 12345,
    "records": [
        {
            "fields": [
                {
                    "name": "ID",
```

```
                    "value": "CU00048376"
                },
                {
                    "name": "Gender",
                    "value": "M."
                },
                {
                    "name": "FirstName",
                    "value": "Benjamin"
                },
                {
                    "name": "LastName",
                    "value": "Verret"
                }
            ]
        },
        {
            "fields": [
                {
                    "name": "ID",
                    "value": "CU01499303"
                },
                {
                    "name": "Gender",
                    "value": "Miss"
                },
                {
                    "name": "FirstName",
                    "value": "Dianne"
                },
                {
                    "name": "LastName",
                    "value": "Straka"
                }
            ]
        }
    ]
}
```

```
{
    "recordid": 12345,
    "table": "detail",
    "records": [
        {
            "fields": [
                {
                    "name": "ItemNumber",
                    "value": "PSM002"
                },
                {
                    "name": "ItemDesc",
                    "value": "PSM Production (unlimited)"
                },
                {
                    "name": "ItemUnitPrice",
                    "value": "495.00"
                },
                {
                    "name": "ItemOrdered",
                    "value": "2"
                },
                {
                    "name": "ItemTotal",
                    "value": "990.00"
                }
            ]
        },
        {
            "fields": [
                {
                    "name": "ItemNumber",
                    "value": "PSM005"
                },
                {
                    "name": "ItemDesc",
                    "value": "Upgrade (Starter to Web)"
                },
```

```
                {
                    "name": "ItemUnitPrice",
                    "value": "495.00"
                },
                {
                    "name": "ItemOrdered",
                    "value": "1"
                },
                {
                    "name": "ItemTotal",
                    "value": "495.00"
                }
            ]
        }
    ]
}
```

JSON New Record Lists

Describes multiple lists of new data records (and their data field values (as name/value pairs)) to be added as data record entities to either existing data set or data record entities of a specific ID.

Structure

The structure consists of an `array` of JSON New Record List structure `objects`.

Example

The following is an example of this structure:

```
[
    {
        "datasetid": 12345,
        "records": [
            {
                "fields": [
                    {
                        "name": "ID",
                        "value": "CU00048376"
                    },
                    {
                        "name": "Gender",
                        "value": "M."
                    },
                    {
                        "name": "FirstName",
                        "value": "Benjamin"
                    },
                    {
                        "name": "LastName",
                        "value": "Verret"
                    }
                ]
            },
            {
                "fields": [
                    {
                        "name": "ID",
```

```
                    "value": "CU01499303"
                },
                {
                    "name": "Gender",
                    "value": "Miss"
                },
                {
                    "name": "FirstName",
                    "value": "Dianne"
                },
                {
                    "name": "LastName",
                    "value": "Straka"
                }
            ]
        }
    ]
},
{
    "recordid": 12345,
    "table": "detail",
    "records": [
        {
            "fields": [
                {
                    "name": "ItemNumber",
                    "value": "PSM002"
                },
                {
                    "name": "ItemDesc",
                    "value": "PSM Production (unlimited)"
                },
                {
                    "name": "ItemUnitPrice",
                    "value": "495.00"
                },
                {
                    "name": "ItemOrdered",
```

```
                        "value": "2"
                    },
                    {
                        "name": "ItemTotal",
                        "value": "990.00"
                    }
                ]
            },
            {
                "fields": [
                    {
                        "name": "ItemNumber",
                        "value": "PSM005"
                    },
                    {
                        "name": "ItemDesc",
                        "value": "Upgrade (Starter to Web)"
                    },
                    {
                        "name": "ItemUnitPrice",
                        "value": "495.00"
                    },
                    {
                        "name": "ItemOrdered",
                        "value": "1"
                    },
                    {
                        "name": "ItemTotal",
                        "value": "495.00"
                    }
                ]
            }
        ]
    }
]
```

JSON Operations List

Describes a list of workflow operations (specifically *asynchronous* workflow operations) actively running on the server, each containing various properties including the type of workflow operation, it's starting time and it's current progress value.

This structure is used specifically with *workflow* based services including the Data Mapping, Content Creation, Content Creation (Email), Job Creation, Output Creation and All-In-One services.

> See the Workflow Operations page of the Technical Overview section for further detail on workflow operations.

Structure

The structure consists of an `array` of `objects` each with the following name/value pairs:

- `id` – the workflow operation identifier (*type* of `string`)
- `type` – the workflow operation type (*value* of either `DataMiningRestService`, `ContentCreationRestService`, `EmailExportRestService`, `JobCreationRestService`, `OutputCreationRestService` or `PrintRestService`) (*type* of `string`)
- `subTask` – the workflow operation sub-task name (*type* of `string`)
- `startTime` – the workflow operation starting time stamp (*value* of milliseconds since midnight of January 1, 1970 UTC) (*type* of `number`)
- `progress` – the workflow operation progress percentage (*value* in range of `0` to `100`) (*type* of `number`)

Workflow operation `objects` with a `type` *value* of either `ContentCreationRestService` or `PrintRestService` (usually with a `subTask` *value* of `Content Creation`) can also contain the following name/value pair:

- `template` – the name of the template being used for content creation (*type* of `string`)

Example

The following is an example of this structure:

```
[
    {
        "id": "1281ef9d-7a74-4448-9adf-175a0166f32e",
        "type": "DataMiningRestService",
        "subTask": "Extracting data 25%",
        "startTime": 1482367446908,
```

```
    "progress": 100
},
{
    "id": "b72e2da5-39ea-48de-85cf-a2be321a71bd",
    "type": "ContentCreationRestService",
    "subTask": "Content Creation",
    "startTime": 1482367988332,
    "progress": 12,
    "template": "business-card-ol"
},
{
    "id": "134f55a5-85f5-41d5-a0d3-e033eda45cb5",
    "type": "EmailExportRestService",
    "startTime": 1482368638197,
    "progress": 5
},
{
    "id": "d52cf2b6-9ca7-44e6-b548-5b249dedf40d",
    "type": "JobCreationRestService",
    "subTask": "Job Creation",
    "startTime": 1482367723483,
    "progress": 77
},
{
    "id": "02fa495b-ed56-47ef-ac49-e63df298b10e",
    "type": "OutputCreationRestService",
    "subTask": "Output Creation",
    "startTime": 1482367851340,
    "progress": 34
},
{
    "id": "fb414be9-4ec5-463a-8429-93153db73783",
    "type": "PrintRestService",
    "subTask": "Content Creation",
    "startTime": 1482366891203,
    "progress": 65,
    "template": "letter-ol"
}
```

```
]
```

JSON Page Details List

Describes a list of the page details and identifiers for each content item contained within a specific content set entity.

Page details include the number of pages per media type, along with media specific properties including the name, size, width and height. Used specifically with the Content Set Entity service.

Structure

The structure consists of an `array` of `objects` each with the following name/value pairs:

- `id` – the content item entity identifier (*type* of `number`)
- `pages` – a list of the pages per media, consisting of an `array` of `objects` each with the following name/value pairs:
    - `count` – the number of pages using the specific media (*type* of `number`)
    - `media` – media specific properties, consisting of an `object` with the following name/value pairs:
        - `name` – the name of the media (*type* of `string`)
        - `size` – the size of the media (*type* of `string`)
        - `width` – the width of the media (*type* of `string`)
        - `height` – the height of the media (*type* of `string`)

Example

The following is an example of this structure:

```
[
    {
        "id": 12345,
        "pages": [
            {
                "count": 2,
                "media": {
                    "name": "Plain A4 Paper",
                    "size": "A4",
                    "width": "210mm",
                    "height": "297mm"
                }
            },
```

```
            {
                "count": 1,
                "media": {
                    "name": "Plain Letter Paper",
                    "size": "Letter",
                    "width": "8.5in",
                    "height": "11in"
                }
            }
        ]
    },
    {
        "id": 23456,
        "pages": [
            {
                "count": 2,
                "media": {
                    "name": "Plain A4 Paper",
                    "size": "A4",
                    "width": "210mm",
                    "height": "297mm"
                }
            },
            {
                "count": 2,
                "media": {
                    "name": "Plain Letter Paper",
                    "size": "Letter",
                    "width": "8.5in",
                    "height": "11in"
                }
            }
        ]
    }
]
```

JSON Page Details Summary

Describes a summary of the page details for a specific content set entity.

Page details include the number of pages per media type, along with media specific properties including the name, size, width and height. Used specifically with the Content Set Entity service.

Structure

The structure consists of an `object` with the following name/value pairs:

- `pages` – a list of the total pages per media, consisting of an `array` of `objects` each with the following name/value pairs:
  - `count` – the number of pages using the specific media (*type* of `number`)
  - `media` – media specific properties, consisting of an `object` with the following name/value pairs:
    - `name` – the name of the media (*type* of `string`)
    - `size` – the size of the media (*type* of `string`)
    - `width` – the width of the media (*type* of `string`)
    - `height` – the height of the media (*type* of `string`)

Example

The following is an example of this structure:

```
{
    "pages": [
        {
            "count": 200,
            "media": {
                "name": "Plain A4 Paper",
                "size": "A4",
                "width": "210mm",
                "height": "297mm"
            }
        },
        {
            "count": 108,
            "media": {
                "name": "Plain Letter Paper",
```

```
                "size": "Letter",
                "width": "8.5in",
                "height": "11in"
            }
        }
    ]
}
```

JSON Parameters

Describes a list of parameters used specifically in the creation of content.

Structure

The structure consists of an `object` with the following name/value pairs:

- `parameters` – a set of the runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

Example

The following is an example of this structure:

```
{
    "parameters": {
           "country": "Canada",
           "InvoiceOverdue": true
    }
}
```

JSON Record Content List

Describes a list of data fields (as name/value pairs), nested data records (if any), along with a number of additional properties for a data record entity of a specific ID.

Tip: A data record entity (in the root or *master* data table) can contain one or more data tables that each contain one or more data record entities (*nested* data record entities).

A nested data record entity can itself contain one or more data tables that each contain one or more nested data record entities, and so on for potentially multiple levels of nested data tables and data record entities.

A data record entity that contains a data table of nested data record entities is considered to be the *parent* of the data record entities contained in that data table (which are considered to be the *children*).

See the "Data Entities" on page 29 page of the "Technical Overview" on page 12 section for further detail on data set and data record entities.

Note: Date/time values in this JSON structure are written in the following format: YYYY-MM-DDTHH:MM:SSZ.

Structure

The structure consists of an `object` with the following name/value pairs:

- `id` – the data record entity identifier (*type* of `number`)
- `fields` – a list of data fields in the data record entity, consisting of an `array` of `objects` each with the following name/value pairs:
  - `name` – the name of the data field (*type* of `string`)
  - `value` – the value of the data field (*type* of `string`)
- `records` – a list of any nested data record entities, consisting of an `array` of `objects` each with the following name/value pairs:
  - `id` – the data record entity identifier (*type* of `number`)
  - `table` – the data record entity data table name (*type* of `string`)
  - `parentrecordid` – the data record entity identifier of parent entity (*type* of `number`)
  - `fields` – a list of data fields in the data record entity, consisting of an `array` of `objects` each with the following name/value pairs:

- `name` – the name of the data field (*type* of `string`)
- `value` – the value of the data field (*type* of `string`)

Specific to *data record* entities that are children of a *data set* entity (data record entities in the root or *master* data table), two additional name/value pairs are included:

- `table` – the data record entity data table name (*value* of `record`) (*type* of `string`)
- `datasetid` – the data set entity identifier of parent entity (*type* of `number`)

If a *data record* entity contains boundary information (set from the data source during data mapping), then an additional name/value pair is also included:

- `boundaries` – the boundaries for the data record, consisting of an `object` with the following name/value pairs:
  - `start` – the starting boundary value for the data record (*type* of `number`)
  - `end` – the ending boundary value for the data record (*type* of `number`)

Specific to *nested data record* entities that are children of a *data record* entity, two additional name/-value pairs are included:

- `table` – the data record entity data table name (*type* of `string`)
- `parentrecordid` – the data record entity identifier of parent entity (*type* of `number`)

Example

The following are examples of this structure:

```
{
    "id": 12345,
    "table": "record",
    "datasetid": 34567,
    "fields": [
        {
            "name": "ID",
            "value": "CU00048376"
        },
        {
            "name": "Gender",
            "value": "M."
        },
        {
```

```
            "name": "FirstName",
            "value": "Benjamin"
        },
        {
            "name": "LastName",
            "value": "Verret"
        }
    ]
}

{
    "id": 45678,
    "table": "detail",
    "parentrecordid": 23456,
    "fields": [
        {
            "name": "ItemNumber",
            "value": "PSM002"
        },
        {
            "name": "ItemDesc",
            "value": "PSM Production (unlimited)"
        },
        {
            "name": "ItemUnitPrice",
            "value": "495.00"
        },
        {
            "name": "ItemOrdered",
            "value": "2"
        },
        {
            "name": "ItemTotal",
            "value": "990.00"
        }
    ]
}
```

```
{
    "id": 23456,
    "table": "record",
    "datasetid": 12345,
    "fields": [
        {
            "name": "ID",
            "value": "CU00048376"
        },
        {
            "name": "Date",
            "value": "2012-03-29T13:00Z"
        },
        {
            "name": "DueDate",
            "value": "2012-04-28T14:00Z"
        },
        {
            "name": "InvNumber",
            "value": "INV9441991"
        },
        {
            "name": "Gender",
            "value": "M."
        },
        {
            "name": "FirstName",
            "value": "Benjamin"
        },
        {
            "name": "LastName",
            "value": "Verret"
        }
        {
            "name": "TotalOrdered",
            "value": "3"
        },
        {
```

```
            "name": "InvSubTotal",
            "value": "1485.00"
        },
        {
            "name": "InvTaxTotal",
            "value": "111.38"
        },
        {
            "name": "InvTotal",
            "value": "1596.38"
        }
    ],
    "records": [
        {
            "id": 45678,
            "table": "detail",
            "parentrecordid": 23456,
            "fields": [
                {
                    "name": "ItemNumber",
                    "value": "PSM002"
                },
                {
                    "name": "ItemDesc",
                    "value": "PSM Production (unlimited)"
                },
                {
                    "name": "ItemUnitPrice",
                    "value": "495.00"
                },
                {
                    "name": "ItemOrdered",
                    "value": "2"
                },
                {
                    "name": "ItemTotal",
                    "value": "990.00"
                }
```

```
            ]
        },
        {
            "id": 45679,
            "table": "detail",
            "parentrecordid": 23456,
            "fields": [
                {
                    "name": "ItemNumber",
                    "value": "PSM005"
                },
                {
                    "name": "ItemDesc",
                    "value": "Upgrade (Starter to Web)"
                },
                {
                    "name": "ItemUnitPrice",
                    "value": "495.00"
                },
                {
                    "name": "ItemOrdered",
                    "value": "1"
                }
                {
                    "name": "ItemTotal",
                    "value": "495.00"
                }
            ]
        }
    ]
}

{
    "id": 12345,
    "table": "record",
    "boundaries": {
        "start": 0,
        "end": 4
    },
```

```
    "datasetid": 34567,
    "fields": [
        {
            "name": "ID",
            "value": "CU00048376"
        },
        {
            "name": "Gender",
            "value": "M."
        },
        {
            "name": "FirstName",
            "value": "Benjamin"
        },
        {
            "name": "LastName",
            "value": "Verret"
        }
    ]
}
```

JSON Record Content List (Explicit Types)

Describes a list of data fields (as name/value pairs), a data table schema, nested data records (if any), along with a number of additional properties for a data record entity of a specific ID.

Unlike a JSON Record Content List structure, this structure includes a data table schema (of data column/field data types) and uses specific JSON types to represent the value of data fields in the data record.

Tip: A data record entity (in the root or *master* data table) can contain one or more data tables that each contain one or more data record entities (*nested* data record entities).

A nested data record entity can itself contain one or more data tables that each contain one or more nested data record entities, and so on for potentially multiple levels of nested data tables and data record entities.

A data record entity that contains a data table of nested data record entities is considered to be the *parent* of the data record entities contained in that data table (which are considered to be the *children*).

See the "Data Entities" on page 29 page of the "Technical Overview" on page 12 section for further detail on data set and data record entities.

Note: Date/time values are stored in the OL Connect database as Unix timestamps, i.e. the total number of seconds from the starting time of UTC (Universal Time, Coordinated) to the current time (with zero time zone offset). For instance: 1675950179.

In this JSON structure, dates are represented the same way.

Structure

The structure consists of an `object` with the following name/value pairs:

- `id` – the data record entity identifier (*type* of `number`)
- `schema` – the data table schema for the data record entity, consisting of an `object` with the following name/value pairs:
  - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an `object` with one or more name/value pairs:
    - *`<name>`* – the name (*name*) and data type of the data field (*value* of either `BOOLEAN`, `STRING`, `HTMLSTRING`, `INTEGER`, `FLOAT`, `DATETIME` or `CURRENCY`) (*type* of `string`)

        - `tables` – a list of any nested data tables in the data record entity, consisting of an `object` with one or more name/value pairs:
            - *`<name>`* – the name (*name*) of the data table and the data table schema for the data record entities it contains, consisting of an `object` with the following name/value pair:
                - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an `object` with one or more name/-value pairs:
                    - *`<name>`* – the name (*name*) and data type of the data field (*value* of either `BOOLEAN`, `STRING`, `HTMLSTRING`, `INTEGER`, `FLOAT`, `DATETIME` or `CURRENCY`) (*type* of `string`)
- `fields` – a list of the data fields in the data record entity and their corresponding data values, consisting of an `object` with one or more name/value pairs:
    - *`<name>`* – the name (*name*) and data value of the data field (*type* of either `string`, `number`, or `boolean`)
- `tables` – a list of any nested data tables in the data record entity, consisting of an `object` with one or more name/value pairs:
    - *`<name>`* – the name (*name*) of the data table and a list the data record entities it contains, consisting of an `array` of `objects` each with the following name/value pairs:
        - `id` – the data record entity identifier (*type* of `number`)
        - `fields` – a list of the data fields in the data record entity and their corresponding data values, consisting of an `object` with one or more name/value pairs:
            - *`<name>`* – the name (*name*) and data value of the data field (*type* of either `string`, `number`, or `boolean`)

Specific to *data record* entities that are children of a *data set* entity (data record entities in the root or *master* data table), an additional name/value pair is included:

- `datasetid` – the data set entity identifier of parent entity (*type* of `number`)

If a *data record* entity contains boundary information (set from the data source during data mapping), then an additional name/value pair is also included:

- `boundaries` – the boundaries for the data record, consisting of an `object` with the following name/value pairs:

- `start` – the starting boundary value for the data record (*type* of `number`)
- `end` – the ending boundary value for the data record (*type* of `number`)

Example

The following are examples of this structure:

```
{
    "schema": {
        "columns": {
            "ID": "STRING",
            "Gender": "STRING",
            "FirstName": "STRING",
            "LastName": "STRING",
            "ExtraData": "STRING"
        }
    },
    "id": 12345,
    "datasetid": 34567,
    "fields": {
        "ID": "CU00048376",
        "Gender": "M.",
        "FirstName": "Benjamin",
        "LastName": "Verret",
        "ExtraData": ""
    }
}

{
    "schema": {
        "columns": {
            "ItemNumber": "STRING",
            "ItemDesc": "STRING",
            "ItemUnitPrice": "CURRENCY",
            "ItemOrdered": "INTEGER",
            "ItemTotal": "CURRENCY",
            "ExtraData": "STRING"
        }
    },
    "id": 45678,
```

```
    "fields": {
        "ItemNumber": "PSM002",
        "ItemDesc": "PSM Production (unlimited)",
        "ItemUnitPrice": "495.00",
        "ItemOrdered": 2,
        "ItemTotal": "990.00",
        "ExtraData": ""
    }
}

{
    "schema": {
        "columns": {
            "ID": "STRING",
            "Date": "DATETIME",
            "DueDate": "DATETIME",
            "InvNumber": "STRING",
            "Gender": "STRING",
            "FirstName": "STRING",
            "LastName": "STRING",
            "TotalOrdered": "INTEGER",
            "InvSubTotal": "CURRENCY",
            "InvTaxTotal": "CURRENCY",
            "InvTotal": "CURRENCY",
            "ExtraData": "STRING"
        },
        "tables": {
            "detail": {
                "columns": {
                    "ItemNumber": "STRING",
                    "ItemDesc": "STRING",
                    "ItemUnitPrice": "CURRENCY",
                    "ItemOrdered": "INTEGER",
                    "ItemTotal": "CURRENCY",
                    "ExtraData": "STRING"
                }
            }
        }
    },
```

```
"id": 23456,
"datasetid": 12345,
"fields": {
    "ID": "CU00048376",
    "Date": "2023-01-26T16:47:35Z",
    "DueDate": "2023-02-23T14:00:00Z",
    "InvNumber": "INV9441991",
    "Gender": "M.",
    "FirstName": "Benjamin",
    "LastName": "Verret",
    "TotalOrdered": 3,
    "InvSubTotal": "1485.00",
    "InvTaxTotal": "111.38",
    "InvTotal": "1596.38",
    "ExtraData": ""
},
"tables": {
    "detail": [
        {
            "id": 45678,
            "fields": {
                "ItemNumber": "PSM002",
                "ItemDesc": "PSM Production (unlimited)",
                "ItemUnitPrice": "495.00",
                "ItemOrdered": 2,
                "ItemTotal": "990.00",
                "ExtraData": ""
            }
        },
        {
            "id": 45679,
            "fields": {
                "ItemNumber": "PSM005",
                "ItemDesc": "Upgrade (Starter to Web)",
                "ItemUnitPrice": "495.00",
                "ItemOrdered": 1,
                "ItemTotal": "495.00",
                "ExtraData": ""
```

```
                }
            }
        ]
    }
}

{
    "schema": {
        "columns": {
            "ID": "STRING",
            "Gender": "STRING",
            "FirstName": "STRING",
            "LastName": "STRING",
            "ExtraData": "STRING"
        }
    },
    "id": 12345,
    "datasetid": 34567,
    "boundaries": {
        "start": 0,
        "end": 4
    },
    "fields": {
        "ID": "CU00048376",
        "Gender": "M.",
        "FirstName": "Benjamin",
        "LastName": "Verret",
        "ExtraData": ""
    }
}
```

JSON Record Content List (Fields Only)

Describes a list of data field values (as name/value pairs) for a data record, used to update an existing data record entity of a specific ID.

> **Note:** Date/time values are stored in the OL Connect database as Unix timestamps, i.e. the total number of seconds from the starting time of UTC (Universal Time, Coordinated) to the current time (with zero time zone offset). For instance: 1675950179.
>
> In this JSON structure, dates are represented the same way.

Structure

The structure consists of an `object` with the following name/value pairs:

- `id` – the data record entity identifier (*type* of `number`)
- `fields` – a list of data fields in the data record entity, consisting of an `array` of `objects` each with the following name/value pairs:
  - `name` – the name of the data field (*type* of `string`)
  - `value` – the value of the data field (*type* of `string`)

Example

The following is an example of this structure:

```
{
    "id": 12345,
    "fields": [
        {
            "name": "FirstName",
            "value": "Benjamin"
        },
        {
            "name": "LastName",
            "value": "Verret"
        }
    ]
}
```

JSON Record Content Lists

Describes multiple lists of data field values (as name/value pairs), nested data records (if any), along with a number of additional properties for data record entities of a specific ID.

Structure

The structure consists of an `array` of JSON Record Content List structure `objects`.

Example

The following is an example of this structure:

```
[
    {
        "id": 45678,
        "table": "detail",
        "parentrecordid": 23456,
        "fields": [
            {
                "name": "ItemNumber",
                "value": "PSM002"
            },
            {
                "name": "ItemDesc",
                "value": "PSM Production (unlimited)"
            },
            {
                "name": "ItemUnitPrice",
                "value": "495.00"
            },
            {
                "name": "ItemOrdered",
                "value": "2"
            },
            {
                "name": "ItemTotal",
                "value": "990.00"
            }
        ]
    },
```

```
    {
        "id": 45679,
        "table": "detail",
        "parentrecordid": 23456,
        "fields": [
            {
                "name": "ItemNumber",
                "value": "PSM005"
            },
            {
                "name": "ItemDesc",
                "value": "Upgrade (Starter to Web)"
            },
            {
                "name": "ItemUnitPrice",
                "value": "495.00"
            },
            {
                "name": "ItemOrdered",
                "value": "1"
            }
            {
                "name": "ItemTotal",
                "value": "495.00"
            }
        ]
    }
]
```

JSON Record Content Lists (Explicit Types)

Describes multiple lists of data field values (as name/value pairs), a data table schema, nested data records (if any), along with a number of additional properties for data record entities of a specific ID.

Structure

The structure consists of an `array` of JSON Record Content List (Explicit Types) structure `objects`.

Example

The following is an example of this structure:

```
[
    {
        "schema": {
            "columns": {
                "ItemNumber": "STRING",
                "ItemDesc": "STRING",
                "ItemUnitPrice": "CURRENCY",
                "ItemOrdered": "INTEGER",
                "ItemTotal": "CURRENCY",
                "ExtraData": "STRING"
            }
        },
        "id": 45678,
        "fields": {
            "ItemNumber": "PSM002",
            "ItemDesc": "PSM Production (unlimited)",
            "ItemUnitPrice": "495.00",
            "ItemOrdered": 2,
            "ItemTotal": "990.00",
            "ExtraData": ""
        }
    },
    {
        "schema": {
            "columns": {
                "ItemNumber": "STRING",
                "ItemDesc": "STRING",
                "ItemUnitPrice": "CURRENCY",
```

```
            "ItemOrdered": "INTEGER",
            "ItemTotal": "CURRENCY",
            "ExtraData": "STRING"
        }
    },
    "id": 45679,
    "fields": {
        "ItemNumber": "PSM005",
        "ItemDesc": "Upgrade (Starter to Web)",
        "ItemUnitPrice": "495.00",
        "ItemOrdered": 1,
        "ItemTotal": "495.00",
        "ExtraData": ""
    }
}
]
```

JSON Record Content Lists (Fields Only)

Describes lists of data field values (as name/value pairs) for multiple data records.

Structure

The structure consists of an `array` of JSON Record Content List (Fields Only) structure `objects`.

Example

The following is an example of this structure:

```
[
    {
        "id": 12345,
        "fields": [
            {
                "name": "FirstName",
                "value": "Benjamin"
            },
            {
                "name": "LastName",
                "value": "Verret"
            }
        ]
    },
    {
        "id": 23456,
        "fields": [
            {
                "name": "FirstName",
                "value": "Dianne"
            },
            {
                "name": "LastName",
                "value": "Straka"
            }
        ]
    }
]
```

JSON Record Data List

Describes a list of data fields (as name/value pairs), a data table schema and nested data records (if any) for one or more data records.

This structure is used specifically by the Content Creation and Content Creation (HTML) services when creating content directly without the prerequisite of the data mapping process.

Structure

The structure consists of an `object` with the following name/value pair:

- `data` – the data for the data record or data records, consisting of either an `object` or an `array` of one or more `objects` respectively with the following name/value pairs:
  - `schema` – the data table schema for the data record, consisting of an `object` with the following name/value pairs:
    - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an `object` with one or more name/value pairs:
      - *`<name>`* – the name (***name***) and data type of the data field (***value*** of either `BOOLEAN`, `STRING`, `HTMLSTRING`, `INTEGER`, `FLOAT`, `DATETIME` or `CURRENCY`) (***type*** of `string`)
    - `tables` – a list of any nested data tables in the data record, consisting of an `object` with one or more name/value pairs:
      - *`<name>`* – the name (***name***) of the data table and the data table schema for the data records it contains, consisting of an `object` with the following name/-value pair:
        - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an `object` with one or more name/value pairs:
          - *`<name>`* – the name (***name***) and data type of the data field (***value*** of either `BOOLEAN`, `STRING`, `HTMLSTRING`, `INTEGER`, `FLOAT`, `DATETIME` or `CURRENCY`) (***type*** of `string`)
  - `fields` – a list of the data fields in the data record and their corresponding data values, consisting of an `object` with one or more name/value pairs:
    - *`<name>`* – the name (***name***) and data value of the data field (***type*** of either `string`, `number`, or `boolean`)

- `tables` – a list of any nested data tables in the data record, consisting of an `object` with one or more name/value pairs:
    - *<name>* – the name (*name*) of the data table and a list the data records it contains, consisting of an `array` of `objects` each with the following name/value pairs:
        - `id` – a required/default fixed *value* of `0` for all data records (*type* of `number`)
        - `fields` – a list of the data fields in the data record and their corresponding data values, consisting of an `object` with one or more name/value pairs:
            - *<name>* – the name (*name*) and data value of the data field (*type* of either `string`, `number`, or `boolean`)

In addition, a subset of the runtime parameters defined in the template can be passed in the following object:

- `parameters` – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

Example

The following are examples of this structure:

```
{
    "data": {
        "schema": {
            "columns": {
                "ID": "STRING",
                "Gender": "STRING",
                "FirstName": "STRING",
                "LastName": "STRING"
            }
        },
        "fields": {
            "ID": "CU00048376",
            "Gender": "M.",
            "FirstName": "Benjamin",
            "LastName": "Verret"
        }
```

```
    }
}

{
    "data": {
        "schema": {
            "columns": {
                "ID": "STRING",
                "Date": "DATETIME",
                "DueDate": "DATETIME",
                "InvNumber": "STRING",
                "Gender": "STRING",
                "FirstName": "STRING",
                "LastName": "STRING",
                "TotalOrdered": "INTEGER",
                "InvSubTotal": "CURRENCY",
                "InvTaxTotal": "CURRENCY",
                "InvTotal": "CURRENCY"
            },
            "tables": {
                "detail": {
                    "columns": {
                        "ItemNumber": "STRING",
                        "ItemDesc": "STRING",
                        "ItemUnitPrice": "CURRENCY",
                        "ItemOrdered": "INTEGER",
                        "ItemTotal": "CURRENCY"
                    }
                }
            }
       },
        "fields": {
            "ID": "CU00048376",
            "Date": 1332594000000,
            "DueDate": 1335189600000,
            "InvNumber": "INV9441991",
            "Gender": "M.",
            "FirstName": "Benjamin",
            "LastName": "Verret",
```

```
            "TotalOrdered": 3,
            "InvSubTotal": "1485.00",
            "InvTaxTotal": "111.38",
            "InvTotal": "1596.38"
        },
        "tables": {
            "detail": [
                {
                    "id": 0,
                    "fields": {
                        "ItemNumber": "PSM002",
                        "ItemDesc": "PSM Production (unlimited)",
                        "ItemUnitPrice": "495.00",
                        "ItemOrdered": 2,
                        "ItemTotal": "990.00"
                    }
                },
                {
                    "id": 0,
                    "fields": {
                        "ItemNumber": "PSM005",
                        "ItemDesc": "Upgrade (Starter to Web)",
                        "ItemUnitPrice": "495.00",
                        "ItemOrdered": 1,
                        "ItemTotal": "495.00"
                    }
                }
            ]
        }
    }
}

{
    "data": [
        {
            "schema": {
                "columns": {
                    "ID": "STRING",
                    "Gender": "STRING",
```

```
                "FirstName": "STRING",
                "LastName": "STRING"
            }
        },
        "fields": {
            "ID": "CU00048376",
            "Gender": "M.",
            "FirstName": "Benjamin",
            "LastName": "Verret"
        }
    },
    {
        "schema": {
            "columns": {
                "ID": "STRING",
                "Gender": "STRING",
                "FirstName": "STRING",
                "LastName": "STRING"
            }
        },
        "fields": {
            "ID": "CU01499303",
            "Gender": "Miss",
            "FirstName": "Dianne",
            "LastName": "Straka"
        }
    }
]
}
```

JSON Record Data List (with Email Parameters)

Describes a list of data fields (as name/value pairs), a data table schema and nested data records (if any) for one or more data records, along with additional parameters used specifically in an content creation operation for email.

This structure is variable, allowing specification for email output to either the file store or directly to a SMTP mail server, with a number of additional parameters.

This structure is used specifically by the Content Creation (Email) service when creating content directly without the prerequisite of the data mapping process.

Structure

The structure initially consists of an object with the following name/value pair:

- `data` – the data for the data record or data records, consisting of either an object or an array of one or more objects respectively with the following name/value pairs:
    - `schema` – the data table schema for the data record, consisting of an object with the following name/value pairs:
        - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an object with one or more name/value pairs:
            - `<name>` – the name (*name*) and data type of the data field (*value* of either BOOLEAN, STRING, HTMLSTRING, INTEGER, FLOAT, DATETIME or CURRENCY) (*type* of `string`)
        - `tables` – a list of any nested data tables in the data record, consisting of an object with one or more name/value pairs:
            - `<name>` – the name (*name*) of the data table and the data table schema for the data records it contains, consisting of an object with the following name/value pair:
                - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an object with one or more name/value pairs:
                    - `<name>` – the name (name) and data type of the data field (value of either BOOLEAN, STRING, HTMLSTRING, INTEGER, FLOAT, DATETIME or CURRENCY) (*type* of `string`)
    - `fields` – a list of the data fields in the data record and their corresponding data values, consisting of an object with one or more name/value pairs:

        - `<name>` – the name (*name*) and data value of the data field (*type* of either `string`, `number` or `boolean`)
    - `tables` – a list of any nested data tables in the data record, consisting of an object with one or more name/value pairs:
        - `<name>` – the name (*name*) of the data table and a list the data records it contains, consisting of an array of objects each with the following name/value pairs:
            - `id` – a required/default fixed value of 0 for all data records (*type* of `number`)
            - `fields` – a list of the data fields in the data record and their corresponding data values, consisting of an object with one or more name/value pairs:
                - `<name>` – the name (*name*) and data value of the data field (*type* of either `string`, `number` or `boolean`)

A subset of the runtime parameters defined in the template can be passed in the following object:

- `parameters` – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
    - `<name>` – the name (name) and the value of the runtime parameter (*type* of either `string`, `number`, or `boolean`)

For either email output directly to a SMTP mail server or email output to the file store, the following optional name/value pairs can be specified:

- `attachPdfPage` – parameter to specify if a PDF file of the Print context should also be created and attached to the email output (*type* of `boolean`)
- `attachWebPage` – parameter to specify if HTML files of the enabled sections (a single section by default) in the Web context should also be created and attached to the email output (*type* of `boolean`)
- `addPlainText` - parameter to specify if a plain text copy should be added to the email (*type* of `boolean`). This parameter overrides any setting in the template.

Specific to email output directly to a SMTP mail server, an additional name/value pair is required:

- `host` – the network address or name of the SMTP mail server through which emails will be sent. If required, a server port value can also be specified (*type* of `string`)

Specific to email output directly to a SMTP mail server, an optional name/value pair can be specified:

- `useAuth` – parameter to specify if authentication is to be used with the mail server (*type* of `boolean`)

Specific to email output directly to a SMTP mail server *with* the useAuth parameter specified to a value of `true`, the following optional name/value pairs can be specified:

- `user` – the user name to authenticate with (*type* of `string`)
- `password` – the password to authenticate with (*type* of `string`)
- `useStartTLS` – parameter to specify if Transport Layer Security (TLS) is to be opportunistically used when sending emails (*type* of `boolean`)

Specific to email output to the File Store, an optional name/value pair can be specified:

- `eml` – parameter to specify if an EML (E-Mail Message) file of the email for each record should be created in the email output (*type* of `boolean`)

Specific to either email output directly to a SMTP mail server or email output to the file store with the eml parameter specified to a value of true, an additional name/value pair is required:

- `sender` – the email address to be shown as the sender in the email output (*type* of `string`)

Specific to either email output directly to a SMTP mail server or email output to the file store with the sender parameter specified, the following optional name/value pairs can be specified:

- `senderName` – the name to be shown as the sender in the email output (*type* of `string`)
- `useSender` – parameter to specify if the sender address will be used as the receiver address for all emails in the output (*type* of `boolean`)

Examples

The following is an example of this structure:

```
{
    "data": {
        "schema": {
            "columns": {
                "ID": "STRING",
                "Gender": "STRING",
                "FirstName": "STRING",
                "LastName": "STRING",
                "Email": "STRING"
            }
        },
        "fields": {
            "ID": "CU00048376",
```

```
            "Gender": "M.",
            "FirstName": "Benjamin",
            "LastName": "Verret",
            "Email": "b.verret@drupa.ol.com.com"
        }
    },
    "attachPdfPage": true,
    "attachWebPage": true,
    "sender": "john.smith@company.com",
    "useSender": true,
    "host": "mail.company.com",
    "useAuth": true,
    "user": "johns",
    "password": "password5",
}
```

```
{
    "data": [
        {
            "schema": {
                "columns": {
                    "ID": "STRING",
                    "Gender": "STRING",
                    "FirstName": "STRING",
                    "LastName": "STRING",
                    "Email": "STRING"
                }
            },
            "fields": {
                "ID": "CU00048376",
                "Gender": "M.",
                "FirstName": "Benjamin",
                "LastName": "Verret",
                "Email": "b.verret@drupa.ol.com.com"
            }
        },
        {
            "schema": {
                "columns": {
```

```
                    "ID": "STRING",
                    "Gender": "STRING",
                    "FirstName": "STRING",
                    "LastName": "STRING",
                    "Email": "STRING"
                }
            },
            "fields": {
                "ID": "CU01499303",
                "Gender": "Miss",
                "FirstName": "Dianne",
                "LastName": "Straka",
                "Email": "d.straka@drupa.ol.com.com"
            }
        }
    ],
    "attachWebPage": true,
    "sender": "jane.smith@company.com",
    "senderName": "Jane Smith",
    "eml": true
}

{
    "data": {
        "schema": {
            "columns": {
                "ID": "STRING",
                "Gender": "STRING",
                "FirstName": "STRING",
                "LastName": "STRING",
                "Email": "STRING"
            }
        },
        "fields": {
            "ID": "CU01499303",
            "Gender": "Miss",
            "FirstName": "Dianne",
            "LastName": "Straka",
            "Email": "d.straka@drupa.ol.com.com"
```

```
        }
    },
    "attachPdfPage": false,
    "attachWebPage": true
}
```

JSON Record Data List (with Image Parameters)

Describes a list of data fields (as name/value pairs), a data table schema and nested data records (if any) for one or more data records, along with additional parameters used specifically in the creation of a preview image of content for print, email or web.

This structure is used specifically by the Content Creation service when creating preview images.

Structure

The structure initially consists of an `object` with the following name/value pair:

- `data` – the data for the data record or data records, consisting of either an `object` or an `array` of one or more `objects` respectively with the following name/value pairs:
    - `schema` – the data table schema for the data record, consisting of an `object` with the following name/value pairs:
        - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an `object` with one or more name/value pairs:
            - `<name>` – the name (*name*) and data type of the data field (*value* of either `BOOLEAN`, `STRING`, `HTMLSTRING`, `INTEGER`, `FLOAT`, `DATETIME` or `CURRENCY`)
              (*type* of `string`)
        - `tables` – a list of any nested data tables in the data record, consisting of an `object` with one or more name/value pairs:
            - `<name>` – the name (*name*) of the data table and the data table schema for the data records it contains, consisting of an `object` with the following name/-value pair:
                - `columns` – a list of the data columns/fields in the data table schema and their corresponding data types, consisting of an `object` with one or more name/value pairs:

- `<name>` – the name (name) and data type of the data field (*value* of either `BOOLEAN`, `STRING`, `HTMLSTRING`, `INTEGER`, `FLOAT`, `DATETIME` or `CURRENCY`)
  (*type* of `string`)

- `fields` – a list of the data fields in the data record and their corresponding data values, consisting of an `object` with one or more name/value pairs:
  - `<name>` – the name (*name*) and data value of the data field
    (*type* of either `string`, `number` or `boolean`)
- `tables` – a list of any nested data tables in the data record, consisting of an `object` with one or more name/value pairs:
  - `<name>` – the name (*name*) of the data table and a list the data records it contains, consisting of an `array` of `objects` each with the following name/value pairs:
    - `id` – a required/default fixed value of 0 for all data records
      (*type* of `number`)
    - `fields` – a list of the data fields in the data record and their corresponding data values, consisting of an object with one or more name/value pairs:
      - `<name>` – the name (*name*) and data value of the data field
        (*type* of either `string`, `number` or `boolean`)

In addition, a subset of the runtime parameters defined in the template can be passed in the following object:

- `parameters` – a set of runtime parameter names and their corresponding values, consisting of an object with one or more name/value pairs:
  - `<name>` – the name (name) and the value of the runtime parameter
    (*type* of either `string`, `number`, or `boolean`)

Specific to the parameters to be used in the creation of a preview image, the following optional name/-value pairs can be specified:

- `context` – the context to be used in the creation of the preview (*value* of either `print`, `email` or `web`)
  (*type* of `string` – *Default value* is determined by the first context in the template)
- `section` – the section to be used within the context specified (as either the `context` parameter, or else the default context of the template)
  (*type* of `string` – *Default value* is determined by the context specified. For the *Print* context this will be all enabled sections. For the *Email* and *Web* contexts this will be the default section)

- `type` – the image type/format to be used in the creation of the preview (*value* of either `jpg`, `jpeg` or `png`)
  (*type* of `string` – *Default value* of `jpg`)
- `dpi` – the target image resolution of the preview in *dots per inch* (DPI)
  (*type* of `number` – *Default value* of `96`)
- `archive` – whether to return the resulting preview as a ZIP file/archive
  (*type* of `boolean` – *Default value* is determined automatically by the number of image files in the preview output)

Specific to parameters with a `type` parameter specified to a value of png, the following optional name/-value pair can be specified:

- `quality` – the image quality of the preview (*value* ranging from `0-100`)
  (*type* of `number` – *Default value* of `100`)

Specific to parameters with a `context` parameter specified to a value of `print`, the following optional name/value pairs can be specified:

- `bleed` – whether to include the bleed area in the preview
  (*type* of `boolean` – *Default value* of `false`)
- `pages` – the page range to be output in the preview
  (*type* of `string` – *Default value* is determined by the value of the `archive` parameter. If the `archive` parameter is specified to `false`, then the default value will be `1`. If the archive parameter is either omitted or specified to a value of `true`, then the default value will be * (all pages))

Specific to parameters with a `context` parameter specified to a value of either `email`, or `web` the following optional name/value pair can be specified:

- `viewPortWidth` – the image width of the preview in *pixels*
  (*type* of `number` – *Default value* of `1024`)

Examples

The following is an example of this structure:

```
{
    "data": {
        "schema": {
            "columns": {
                "ID": "STRING",
                "Gender": "STRING",
                "FirstName": "STRING",
```

```
                "LastName": "STRING"
            }
        },
        "fields": {
            "ID": "CU00048376",
            "Gender": "M.",
            "FirstName": "Benjamin",
            "LastName": "Verret"
        }
    },
    "context": "print",
    "type": "png",
    "dpi": 150,
    "archive": true,
    "bleed": true,
    "pages": "1-2"
}

{
    "data": {
        "schema": {
            "columns": {
                "ID": "STRING",
                "Date": "DATETIME",
                "DueDate": "DATETIME",
                "InvNumber": "STRING",
                "Gender": "STRING",
                "FirstName": "STRING",
                "LastName": "STRING",
                "TotalOrdered": "INTEGER",
                "InvSubTotal": "CURRENCY",
                "InvTaxTotal": "CURRENCY",
                "InvTotal": "CURRENCY"
            },
            "tables": {
                "detail": {
                    "columns": {
                        "ItemNumber": "STRING",
                        "ItemDesc": "STRING",
```

```
                "ItemUnitPrice": "CURRENCY",
                "ItemOrdered": "INTEGER",
                "ItemTotal": "CURRENCY"
            }
        }
    }
},
 "fields": {
    "ID": "CU00048376",
    "Date": 1332594000000,
    "DueDate": 1335189600000,
    "InvNumber": "INV9441991",
    "Gender": "M.",
    "FirstName": "Benjamin",
    "LastName": "Verret",
    "TotalOrdered": 3,
    "InvSubTotal": "1485.00",
    "InvTaxTotal": "111.38",
    "InvTotal": "1596.38"
 },
 "tables": {
    "detail": [
        {
            "id": 0,
            "fields": {
                "ItemNumber": "PSM002",
                "ItemDesc": "PSM Production (unlimited)",
                "ItemUnitPrice": "495.00",
                "ItemOrdered": 2,
                "ItemTotal": "990.00"
            }
        },
        {
            "id": 0,
            "fields": {
                "ItemNumber": "PSM005",
                "ItemDesc": "Upgrade (Starter to Web)",
                "ItemUnitPrice": "495.00",
```

```
                        "ItemOrdered": 1,
                        "ItemTotal": "495.00"
                    }
                }
            ]
        }
    },
    "context": "web",
    "section": "Section 1",
    "type": "jpeg",
    "viewPortWidth": 1024
}

{
    "data": [
        {
            "schema": {
                "columns": {
                    "ID": "STRING",
                    "Gender": "STRING",
                    "FirstName": "STRING",
                    "LastName": "STRING"
                }
            },
            "fields": {
                "ID": "CU00048376",
                "Gender": "M.",
                "FirstName": "Benjamin",
                "LastName": "Verret"
            }
        }
    ],
    "context": "email",
    "section": "Section 2",
    "type": "jpg",
    "quality": 90
}
```

JSON Search Parameters

Describes a set of complex search criteria broken into search, sorting and grouping rules. This structure is used specifically with the Entity service as input to the Find Data Entity resource method.

Search rules can be added to a search rules list and can be used to match data entities based on specific criteria. This rules list also specifies an operator which determines whether all rules or only one rule in the list is required to be matched.

Search rules can be based on data record values, data entity properties, finishing options, document length, template names and whether an entity's identifier is contained or not contained in a list of identifiers.

Rule sets can also be added to the search rules list. Each rule set can contain its own sub list of search rules and its own rule operator. Rule sets can also be added to the search rule list of an existing rule set which allows for the construction of complex nested search criteria.

Sorting rules can be also added to a sort rules list and (depending on the data entity type) can be used to sort data entity entries in the search results by either data record values or data entity properties.

Every sort rule added will expand the value of the sort key of each entry listed in the resulting JSON Identifier Lists (with Sort Key) structure.

Grouping rules can be also added to a group rules list and (depending on the data entity type) can be used to group data entity entries in the search results by either data record values or data entity properties.

Every group rule added can expand the number of sub lists contained in the resulting JSON Identifier Lists (with Sort Key) structure.

> Certain *search*, *sorting* or *grouping* rules can only be used with specific data entity types.
>
> See the Finding Specific Data Entities in the Server page of the Working Examples section for further detail on the available rule combinations.

Structure

The structure consists of an `object` with the following name/value pairs:

- `entity` – the data entity type (*value* of either `DATARECORDS`, `DATASETS`, `CONTENTITEMS`, `CONTENTSETS`, `JOBS` or `JOBSETS`)
- `search` – search criteria, consisting of an `object` with the following name/value pairs:
  - `ruleType` – the topmost `RULESET`
  - `condition` – the logic rule for this `RULESET` (*value* of `ALL`, `ANY`, `NOTALL` or `NOTANY`)

    - `rules` – a base list of *search* rules, consisting of an `array` of `objects` each with a specific rule sub-structure depending on the type of rule. This could include nested rule sets.
- `sort` – a list of *sorting* rules, consisting of an `array` of `objects` each with the following name/-value pairs:
    - `type` – the type of sorting rule (*value* of either `value` or `property`)
    - `name` – the name of the data value field or data entity property to sort by (*type* of `string`)
    - `numeric` – whether the data value field is a of a numeric type (*type* of `boolean`)
      (only available when sorting on `value` fields)
    - `order` – the order that matches to this rule are sorted by (*value* of either `ASC` or `DESC`)

- `group` – a list of *grouping* rules, consisting of an `array` of `objects` each with the following name/value pairs:
    - `type` – the type of grouping rule (*value* of either `value` or `property`)
    - `name` – the name of the data value field or data entity property to group by (*type* of `string`)
    - `numeric` – whether the data value field is a of a numeric type (*type* of `boolean`)
      (only available when grouping on `value` fields)
    - `order` – the order that matches to this rule are grouped by (*value* of either `ASC` or `DESC`)

The search rule sub-structure consists of an `object` with rule specific groupings of name/value pairs. These include the following:

Data Value search – Search for data entities based on the value of a data record field.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `VALUE`
- `fieldName` – the name of the data record field (*type* of `string`)
- `condition` – the comparison condition (*value* of either `EQ` (=), `NE` (!=), `LT` (<), `GT` (>), `LTE` (<=), `GTE` (>=), `CONTAINS`, `NOT_CONTAINS`, `STARTS_WITH`, `ENDS_WITH`, `LIKE`, `NOT_LIKE`, `IN` or `NOT IN`)
- `value1` – the comparison value
  Can be one of either of the following, depending upon **data field** selection:
    - For **data field name**: These comprise `objects` containing the following name/-value pairs:

        - `type` – `FIELD`
        - `value` – the comparison value (*type* of `string`)
    - For **data field value**: `value1` – the comparison value (*type* of `string`)

Property Value search – Search for data entities based on the value of a data entity property
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `PROPERTY`
- `property` – the name of the data entity property (*type* of `string`)
- `condition` – the comparison condition (*value* of either `EQ` (=), `NE` (!=), `LT` (<), `GT` (>), `LTE` (<=), `GTE` (>=), `CONTAINS`, `NOT_CONTAINS`, `STARTS_WITH`, `ENDS_WITH`, `LIKE`, `NOT_LIKE`, `IN` or `NOT IN`)
- `value` – the comparison value
  Can be one of either of the following, depending upon **property type** selection:
    - For **property name**: These comprise `objects` containing the following name/value pairs:
        - `type` – `FIELD`
        - `value` – the comparison value (*type* of `string`)
    - For **property value**: `value` – the comparison value (*type* of `string`)

Value In search – search for the data values contained within a list.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `VALUEIN`
- `field` – the field name (*type* of `string`)
- `dataType` – the data type to search (*value* of either `FIELD` or `PROPERTY`)
- `condition` – the comparison condition (*value* of either `IN` or `NOT_IN`)
- `values` – the list of data entities (*type* of `string`, or `array` of `strings`

ID In search – search for the ID values contained within a list.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `IDIN`
- `condition` – the comparison condition (*value* of either`IN` or `NOT_IN`)

- `values` – the list of IDs (*type* of `number`, or `array` of `numbers`

Document Media search – search on the media used.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `DOCMEDIA`
- `attribute` – the document media attribute being searched for (*value* of either media name (`NAME`), or front/rear (`FRONT_COATING`/`BACK_COATING`) sheet coating)
- `condition` – the comparison condition (*value* of either `CONTAINS` or `NOT_CONTAINS`)

The comparison value. Can be one of either of the following, depending upon the `attribute` selection:

- `name` – the media name (*type* of `string`)
  (only available for `attribute` = `NAME` selections)
- `coating` – the specified media sheet coating (*value* of either `UNSPECIFIED`, `NONE`, `COATED`, `GLOSSY`, `HIGH_GLOSS`, `INKJET`, `MATTE`, `SATIN` or `SEMI_GLOSS`)
  (only available for `attribute` = `FRONT_COATING` or `BACK_COATING` selections)

Document Binding search – search on how the finished documents are bound.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `DOCBINDING`
- `attribute` – the document binding attribute being searched for (*value* of `STYLE`, `SIDE`, `LOCATION`,or `ANGLE`)
- `condition` – the comparison condition (*value* of either `CONTAINS` or `NOT_CONTAINS`)

The comparison value. Can be one of either of the following, depending upon the `attribute` selection:

- `bindingStyle` – the binding style of the media used (*value* of either `NONE`, `DEFAULT`, `STAPLED`, `GLUED`, `STITCHED`, `ADHESIVE`, `SPINETAPING`, `RING`, `WIREDCOMB`, `PLASTICCOMB` or `COIL`)
  (only available for `attribute` = `STYLE` selections)
- `bindingEdge` – the binding edge of the media used (*value* of either `DEFAULT`, `LEFT`, `RIGHT`, `TOP` or `BOTTOM`)
  (only available for `attribute` = `SIDE` selections)

- `bindingType` – the binding type of the media used (*value* of either `DEFAULT`, `SADDLE`, `SIDE` or `CORNER`)
  (only available for `attribute` `LOCATION` selections)
- `bindingAngle` – the binding angle of the media used (*value* of either `DEFAULT`, `VERTICAL`, `HORIZONTAL` or `ANGLE`)
  (only available for `attribute` = `ANGLE` selections)

Document Size search – search on document size.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `DOCSIZE`
- `entity` – the document size attribute being searched for (*value* of `PAGE`, `SHEET`, or `SECTION`)
- `condition` – the comparison condition (*value* of either `EQ` (=), `NE` (!=), `LT` (<), `GT` (>), `LTE` (<=) or `GTE` (>=))
- `value` – the comparison value (*type* of `number`)

Duplex search – search on whether the document contains any duplex sheets.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `DUPLEX`
- `condition` – whether the document contains any duplex sheets or not (*value* of either `"HAS_DUPLEX"` or `SIMPLEX_ONLY`)

Template search – searches based on the name of the template used during Content Creation.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `TEMPLATE`
- `condition` – the comparison condition (*value* of either `EQ` (=) or `NE` (!=))
- `name` – the comparison value (*type* of `string`)

Rule Set searches – construct a set of rules that are evaluated collectively.
Comprises `objects` containing the following name/value pairs:

- `ruleType` – `RULESET`
- `condition` – the logic rule for this `RULESET` (*value* of `ALL`, `ANY`, `NOTALL` or `NOTANY`)

- `rules` – a sub-list of *search* rules, consisting of an `array` of `objects` each with a certain rule sub-structure depending on the type of rule

Example

The following is an example of this structure:

```
{
    "entity": "CONTENTITEMS",
    "search": {
        "ruleType": "RULESET",
        "condition": "ALL",
        "rules": [
            {
                "ruleType": "DUPLEX",
                "condition": "HAS_DUPLEX"
            },
            {
                "ruleType": "TEMPLATE",
                "condition": "EQ",
                "name": "Rural"
            },
            {
                "ruleType": "RULESET",
                "condition": "ALL",
                "rules": [
                    {
                        "ruleType": "DOCMEDIA",
                        "attribute": "NAME",
                        "condition": "CONTAINS",
                        "name": "Impact"
                    },
                    {
                        "ruleType": "DOCMEDIA",
                        "attribute": "FRONT_COATING",
                        "condition": "CONTAINS",
                        "coating": "HIGH_GLOSS"
                    },
                    {
                        "ruleType": "DOCMEDIA",
```

```
            "attribute": "BACK_COATING",
            "condition": "CONTAINS",
            "coating": "SEMI_GLOSS"
        }
    ]
},
{
    "ruleType": "RULESET",
    "condition": "ALL",
    "rules": [
        {
            "ruleType": "DOCBINDING",
            "attribute": "STYLE",
            "condition": "CONTAINS",
            "bindingStyle": "STAPLED"
        },
        {
            "ruleType": "DOCBINDING",
            "attribute": "SIDE",
            "condition": "CONTAINS",
            "bindingEdge": "LEFT"
        },
        {
            "ruleType": "DOCBINDING",
            "attribute": "LOCATION",
            "condition": "CONTAINS",
            "bindingType": "SIDE"
        },
        {
            "ruleType": "DOCBINDING",
            "attribute": "ANGLE",
            "condition": "CONTAINS",
            "bindingAngle": "ANGLE"
        }
    ]
},
{
    "ruleType": "RULESET",
```

```
            "condition": "ALL",
            "rules": [
                {
                    "ruleType": "DOCSIZE",
                    "entity": "PAGE",
                    "condition": "GT",
                    "value": 4
                },
                {
                    "ruleType": "DOCSIZE",
                    "entity": "SHEET",
                    "condition": "GT",
                    "value": 2
                },
                {
                    "ruleType": "DOCSIZE",
                    "entity": "SECTION",
                    "condition": "GT",
                    "value": 1
                }
            ]
        }
    ]
},
"sort": [
    {
        "type": "value",
        "name": "LastName",
        "numeric": false,
        "order": "ASC"
    }
],
"group": [
    {
        "type": "value",
        "name": "Gender",
        "numeric": false,
        "order": "ASC"
```

```
            }
        ]
    }
```

Working Examples

This section provides a number of working examples that demonstrate the use of the various resources and methods available in the PReS Connect REST API.

For help on getting started with the PReS Connect REST API Cookbook and the working examples, see the Getting Started page.

- Server Security & Authentication
- Working with the File Store
- Working with the Entity Services
- Working with the Workflow Services

Getting Started

This guide provides many working examples to help illustrate the correct use of a given API/method. To achieve this, the guide uses HTML5 & JavaScript/jQuery syntax, and thus, some basic experience and knowledge of these technologies is assumed.

- **HTML5**: https://www.w3schools.com/html/
- **JavaScript**: https://www.w3schools.com/js/
- **jQuery**: https://jquery.com/

Help on installing and getting started with the working examples can be found on the Requirements & Installation and Structure of the Working Examples pages.

Important notes on general use of the working examples can be found in the HTML Input Placeholders & Multiple Value Fields and Display of Working Example Results pages.

If you have server security settings enabled on your PReS Connect server then the Using the Working Examples with Server Security page should be read also.

Requirements & Installation

Requirements

To use the PReS Connect REST API Cookbook with Working Examples source you will require the following:

1. A working installation of PReS Connect.
   For information about the OL Connect Server configuration, please refer to the PlanetPress Connect Help.
2. Any modern web browser able to display HTML5.

Any recent version of Mozilla Firefox, Google Chrome, or Microsoft Edge with support for HTML5 should be suitable for running the working examples contained in this guide. Microsoft Internet Explorer 11 may also be suitable in some cases.

If using Internet Explorer, you may find issues when using the working examples with PReS Connect's **Server Security Settings** set to *enabled.*

The working examples use HTML5 Local Storage to facilitate authentication and certain simplicity / ease-of-use (across browser tabs). Depending on how your Internet Explorer security settings are configured, you may experience issues if the security level of your zone is set too high.

Essentially, the security zone needs to have the security option **Userdata persistence** (under **Miscellaneous**) set to *enabled*. Without this option enabled, the working examples will not function correctly when using them with PReS Connect's **Server Security Settings** set to enabled.

After running the Authenticate/Login to Server working example to re-authenticate, you should only need to refresh existing pages in order for the authentication credentials (token) to be picked up. In the case of Internet Explorer, you may need to restart the browser for the changes to be picked up.

If all else fails, disabling of the **Sever Security Settings** in the PReS Connect Server Preferences should avoid issues with running the various examples on Internet Explorer.

Installation

The working examples source comes pre-installed with PReS Connect and can be located in a sub-directory of your existing PReS Connect installation directory.

To locate the source on Windows:

1. Open up **Windows Explorer** and navigate to the PReS Connect installation directory followed by its **plugins** sub-directory.
2. Find the **com.objectiflune.serverengine.rest.gui** directory and navigate to its **www** sub-directory

3. You should now be exploring the following or similar location:

   C:\Program Files\Objectif Lune\OL Connect\plugins\com.objectiflune.serverengine.rest.gui_1.X.XXXXX.XXXXXXXX-XXXX\www

4. The **www** directory contains a **cookbook** sub-directory, which contains all of the working examples source. You should find a directory structure matching that shown on the Structure of the Working Examples page.

Note: You can access the PReS Connect REST API Cookbook with Working Examples source locally by entering the following URL in your web browser:

http://localhost:9340/serverengine/html/cookbook/index.html

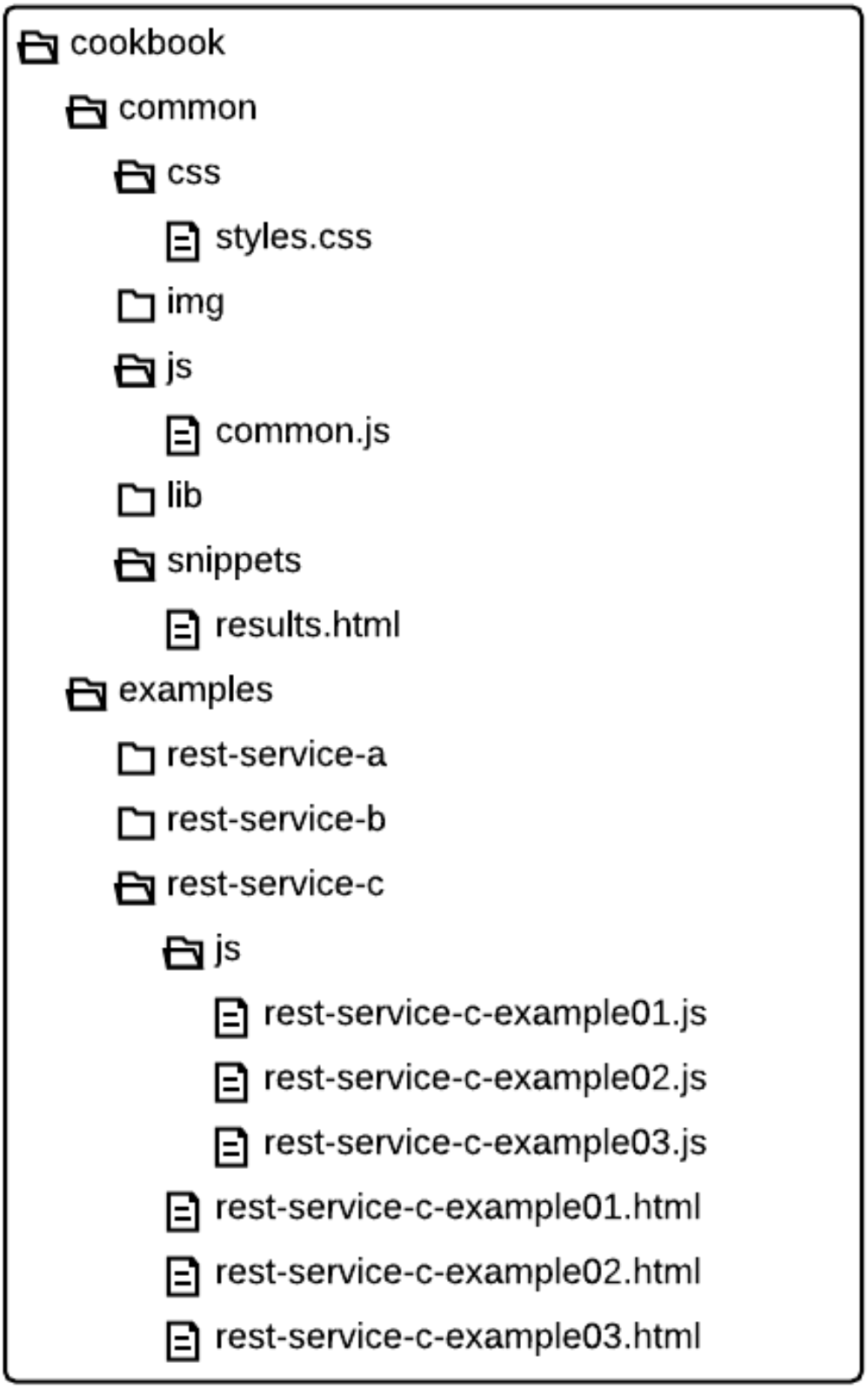
In the event that your Connect Server is configured to use HTTPS instead of HTTP, the following URL must be used:

https://localhost:9340/serverengine/html/cookbook/index.html

Structure of the Working Examples

The working examples are designed to be complete examples, and will generally consists of one HTML5 file paired with a JavaScript/jQuery module which can be found in the *examples/<service-name>/js/* sub-directory.

Where any frequent or boilerplate functionality is commonly used across the examples, this has been moved to the *common/js/common.js* JavaScript/jQuery module.



The examples make use of this module for functionality such as setting up the example, and displaying output results.

The examples also make use of some simple CSS classes as defined in *common/css/styles.css* and HTML snippets for the presentation of output results.

HTML Input Placeholders & Multiple Value Fields

In the working examples, HTML **input** elements make use of the **placeholder** attribute to help provide some indication of the type and format of the value expected to be entered / specified.

The following table lists examples of placeholders commonly used in the working examples:

HTML	Expected Type	Example Values
1234	Single ID Value	• 2341 • 3
1234 or Filename	Single ID or Name Value (File Name)	• 2341 • Promo-EN-1000.csv
1234, 2345, 3456, ...	One or More ID Values (comma separated)	• 2341, 2342 • 3456
Value Username Section Name From Name	String Value	• letter-ol.OL-template • ol-admin • Section 2 • John Smith
Value1, Value2, ...	One or More String Values (comma separated)	• Pablo, Micheal, Maria
Value	Number Value	• 453 • 167.05
dd / mm / yyyy	Date Value	• 22/02/2020
1, 2, 3-5, 6	Numerical Range	• 1, 2, 3 • 1-5 • 1, 2, 3-5, 6
mailbox@domain.com	Email Address Value	• john.smith@contoso.com
mail.domain.com:port	Server Address or Hostname Value (with optional Port)	• smtp.contoso.com • smtp.contoso.com:587 • 192.168.88.54

Display of Working Example Results

When a working example is run, any results will be displayed in a **Results** area that will appear below the working example existing HTML interface.

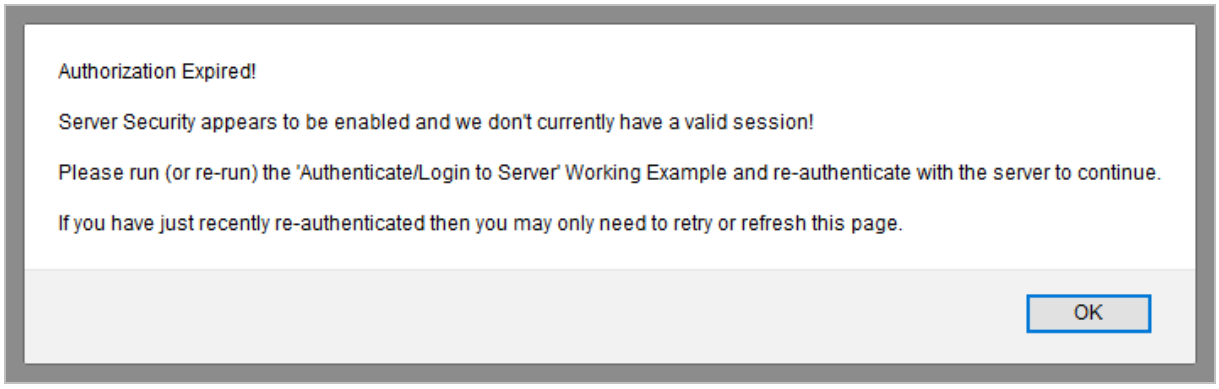
For example:

Note: In some examples the same result will displayed in both *plain* and *JSON structure* based formats. This is to assist ease-of-use when working with outputs of one example that will be needed as an input to another example.

A working example can be run multiple times, and each time the results will be appended below allowing you to compare the output of varying inputs. The **Clear** button can be selected at any time to clear all existing results.

Using the Working Examples with Server Security

If you have the **Server Security Settings** set to *enabled* in your PReS Connect Server Preferences, then you may see the following dialog box initially display when working with the examples:



In the event of this dialog box, just follow the instructions and either refresh the page or re-authenticate by running the Authenticating with the Server (Authenticate/Login to Server) working example covered under the Server Security & Authentication section.

Note: Once re-authenticated, you shouldn't see this dialog box again for as long as your session remains active.

Server Security & Authentication

This section consists of a number of pages covering various useful working examples:

1. Authenticating with the Server

See the Authentication Service page of the REST API Reference section for further detail.

> **Note:** A complete listing including these examples can be found in the **index.html** file located at the root of the working example source code which contains links to all working examples.

Authenticating with the Server

Problem

Your PReS Connect Server is configured to use server security, and you want to authenticate with the server to obtain the correct access to make future requests.

Solution

The solution is to create a request using the following URI and method type to authenticate with the server via the Authentication REST service:

Authenticate/Login to Server	/rest/serverengine/authentication/login	*POST*

Example

HTML5

auth-login-server.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Authenticate/Login to Server Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/auth-login-server.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Authentication Service - Authenticate/Login to Server Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="username">Username:</label>
                    <input id="username" type="text" placeholder="Username" required>
                </div>
                <div>
                    <label for="password">Password:</label>
                    <input id="password" type="password" placeholder="Password" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

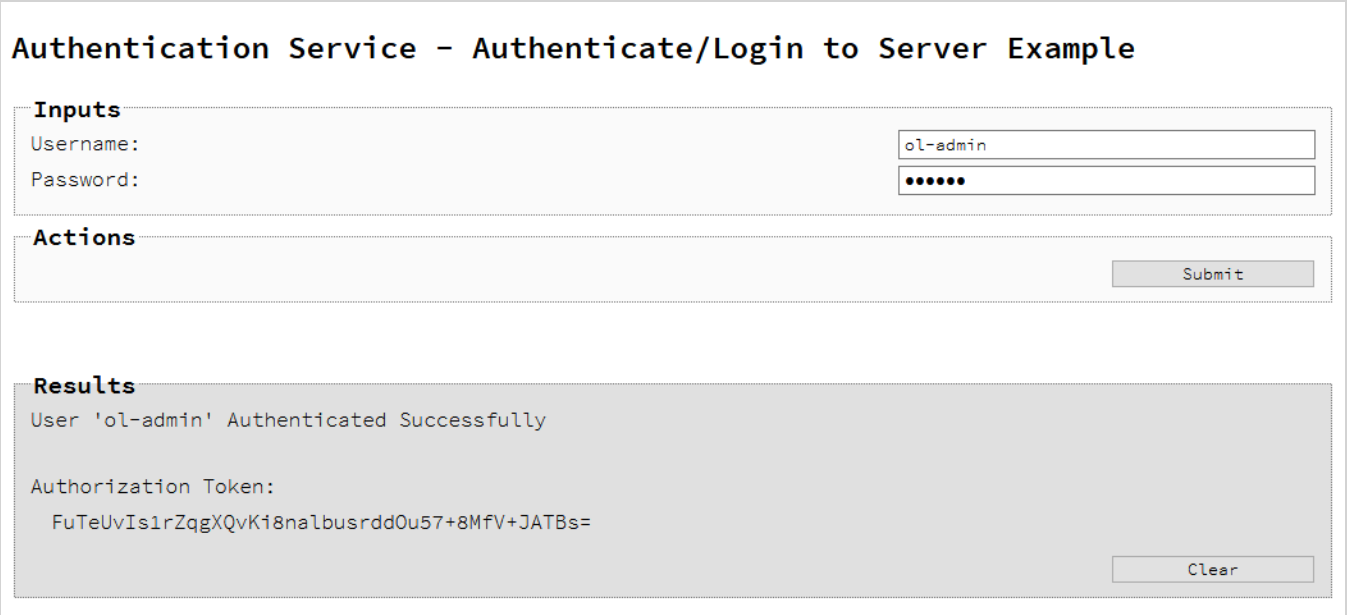
JavaScript/jQuery

auth-login-server.js

```
/* Authentication Service - Authenticate/Login to Server Example */
(function ($, c) {
    "use strict";
    $(function () {
```

```
        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();

            var username = $("#username").val(),
                password = $("#password").val();

            $.ajax({
                type:   "POST",
                url:    "/rest/serverengine/authentication/login",
                beforeSend: function (xhr) {
                    var base64 = "Basic " + btoa(username + ":" + password);
                    xhr.setRequestHeader("Authorization", base64);
                }
            })
                .done(function (response) {
                    c.displayStatus("User '" + username + "' Authenticated Successfully");
                    c.displayResult("Authorization Token", response);
                    c.setSessionToken(response);
                })
                .fail(function (xhr, status, error) {
                    c.displayStatus("Authentication of User '" + username + "' failed!");
                    c.displayResult("Status", xhr.status + " " + error);
                    c.displayResult("Error", xhr.responseText);
                    c.setSessionToken(null);
                });
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter your credentials into the **Username** and **Password** fields and select the **Submit** button.

Once selected, a request containing the credentials will be sent to the server and the result will be returned and displayed to the **Results** area.

If authentication was successful then the response will contain an **Authorization Token** that can be then used in the submission of future requests to the server.

Discussion

Firstly, we define an event handler that will run in response to the submission of the HTML form via the selection of the **Submit** button.

When our event hander function is called, we then obtain the value of the **Username** and **Password** fields. We define two variables, `username` to hold the value of the **Username** text field and `password` to hold the value of the **Password** text field.

Next we construct an jQuery AJAX request which will be sent to the Authentication REST service:

- Method `type` and `url` arguments are specified as shown earlier.
- We specify a `beforeSend` argument containing a function that will add an additional `Authorization` header to the request to facilitate Basic HTTP Authentication. The value of the `Authorization` request header is a Base64 digest of the `username` and `password` variables.

When the request is successful or done, a request response is received and the content of that response is passed as the function parameter `response`. In the example, we then display the value of this parameter which should be the new **Authorization Token** which can then be used in the submission of future requests to the server.

This is achieved by placing the value of the **Authorization Token** in the `auth_token` request header of a future request. In the example the common function `setSessionToken` is used to facilitate this function for all future working example requests.

Further Reading

See the Authentication Service page of the REST API Reference section for further detail.

Working with the File Store

This section consists of a number of pages covering various useful working examples:

1. Uploading a Data File to the File Store
2. Uploading a Data Mapping Configuration to the File Store
3. Uploading a Template to the File Store
4. Uploading a Job Creation Preset to the File Store
5. Uploading an Output Creation Preset to the File Store

See the File Store Service page of the REST API Reference section for further detail.

> **Note:** A complete listing including these examples can be found in the **index.html** file located at the root of the working example source code which contains links to all working examples.

Uploading a Data File to the File Store

Problem

You want to upload a data file to the File Store so that it can be used as part of a Data Mapping operation.

Solution

The solution is to create a request using the following URI and method type to submit the data file to the server via the File Store REST service:

Upload Data File	/rest/serverengine/filestore/DataFile	*POST*

Example

HTML5

fs-datafile-upload.html

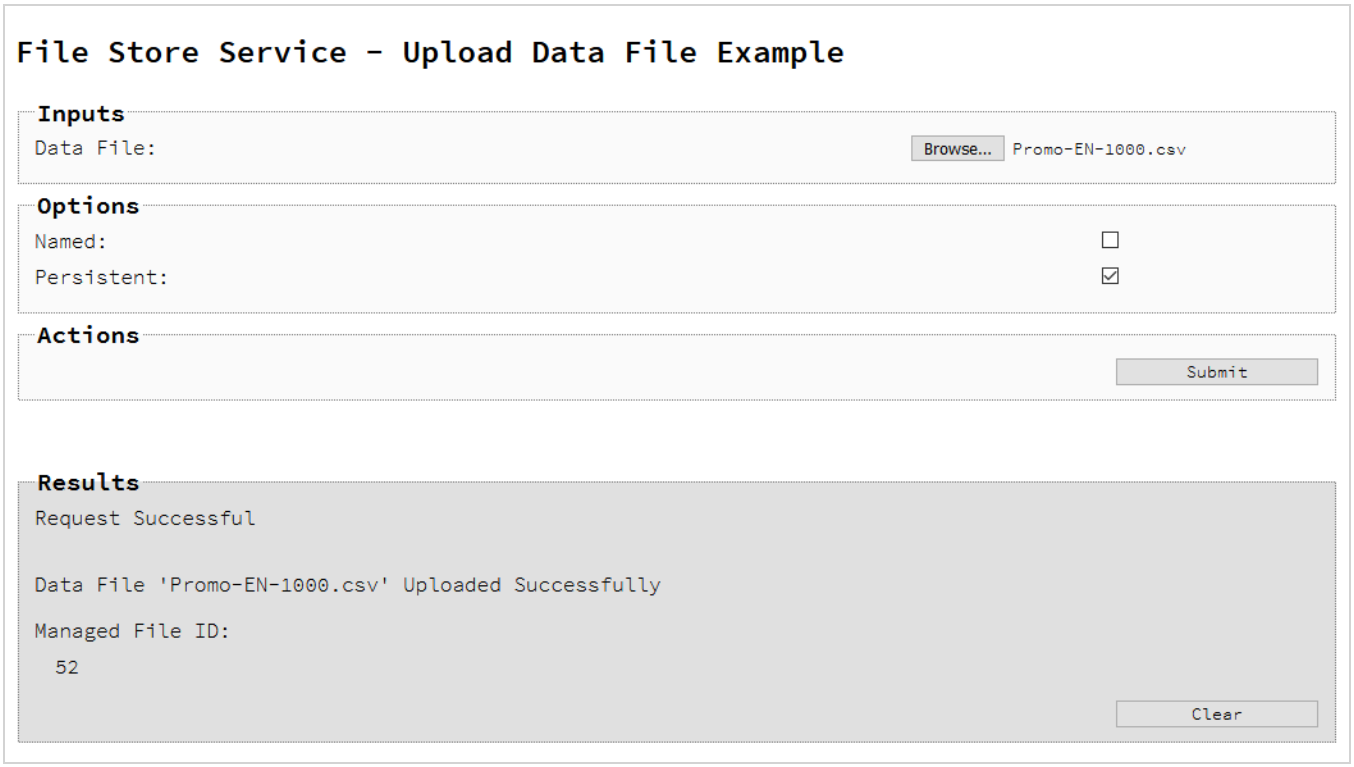
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Upload Data File Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/fs-datafile-upload.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>File Store Service - Upload Data File Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="named">Named:</label>
                    <input id="named" type="checkbox">
                </div>
                <div>
                    <label for="persistent">Persistent:</label>
                    <input id="persistent" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

fs-datafile-upload.js

```
/* File Store Service - Upload Data File Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var file = $("#datafile")[0].files[0],
                named = $("#named").prop("checked"),
                persistent = $("#persistent").prop("checked");

            var settings = {
                type:           "POST",
                url:            "/rest/serverengine/filestore/DataFile?persistent=" + persistent,
                data:           file,
                processData:    false,
                contentType:    "application/octet-stream"
            };
            if (named) settings.url += "&filename=" + file.name;
            $.ajax(settings)
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayInfo("Data File '" + file.name + "' Uploaded Successfully");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply select the **Browse** button and then select the data file you wish to upload using the selection dialog box.

Next you can specify the following options to use with the upload of the data file:

- **Named** – allow this file to be identified/referenced by its Managed File Name as well as its Managed File ID
- **Persistent** – make this file persistent in the file store

> **Note:** Only one Managed File in the file store can be associated with a specific name. If two files are uploaded to the file store under the same name, then only the most recently uploaded file will be associated with (or can be referenced using) that name.

Once the file and options are selected, simply select the **Submit** button to upload the file to the server's file store and the resulting Managed File ID for the data file will be returned and displayed to the **Results** area.

Discussion

Firstly, we define an event handler that will run in response to the submission of the HTML form via the selection of the **Submit** button.

When our event handler function is called, we then obtain a reference to the local data file previously selected. This is achieved by getting the first value of the `files` attribute of the HTML element with the ID of `datafile` (in this case a file type input HTML element) and storing it in a variable `file`.

We also obtain boolean values for the **Named** and **Persistent** options (both checkbox type input HTML elements) and store them in the `named` and `persistent` variables respectively.

Next we construct a jQuery AJAX request which will be sent to the File Store REST service. We use an object called `settings` to hold the arguments for our request:

Method `type` and `url` arguments are specified as shown earlier, with the addition of a `persistent` query parameter which specifies whether the file is to be persistent in the file store when uploaded.

We specify the variable `file` as the `data` or contents of the request, a `contentType` argument of `"application/octet-stream"`, and because we are sending file data we also specify a `processData` argument set to `false`.

If the **Named** option is checked in our form, and the `named` variable is `true`, then a `filename` query parameter is also added which contains the file name of the file selected (`file.name`).

Lastly, the `settings` object is passed as an argument to the jQuery AJAX function `ajax` and the request is executed.

When the request is successful or done, a request response is received and the content of that response is passed as the function parameter `response`. In the example, we then display the value of this parameter which should be the new Managed File ID of the data file in the file store.

Further Reading

See the File Store Service page of the REST API Reference section for further detail.

Uploading a Data Mapping Configuration to the File Store

Problem

You want to upload a data mapping configuration to the File Store so that it can be used as part of a Data Mapping operation.

Solution

The solution is to create a request using the following URI and method type to submit the data mapping configuration to the server via the File Store REST service:

Upload Data Mapping Configuration	/rest/serverengine/filestore/DataMiningConfig	*POST*

Example

HTML5

fs-datamapper-upload.html

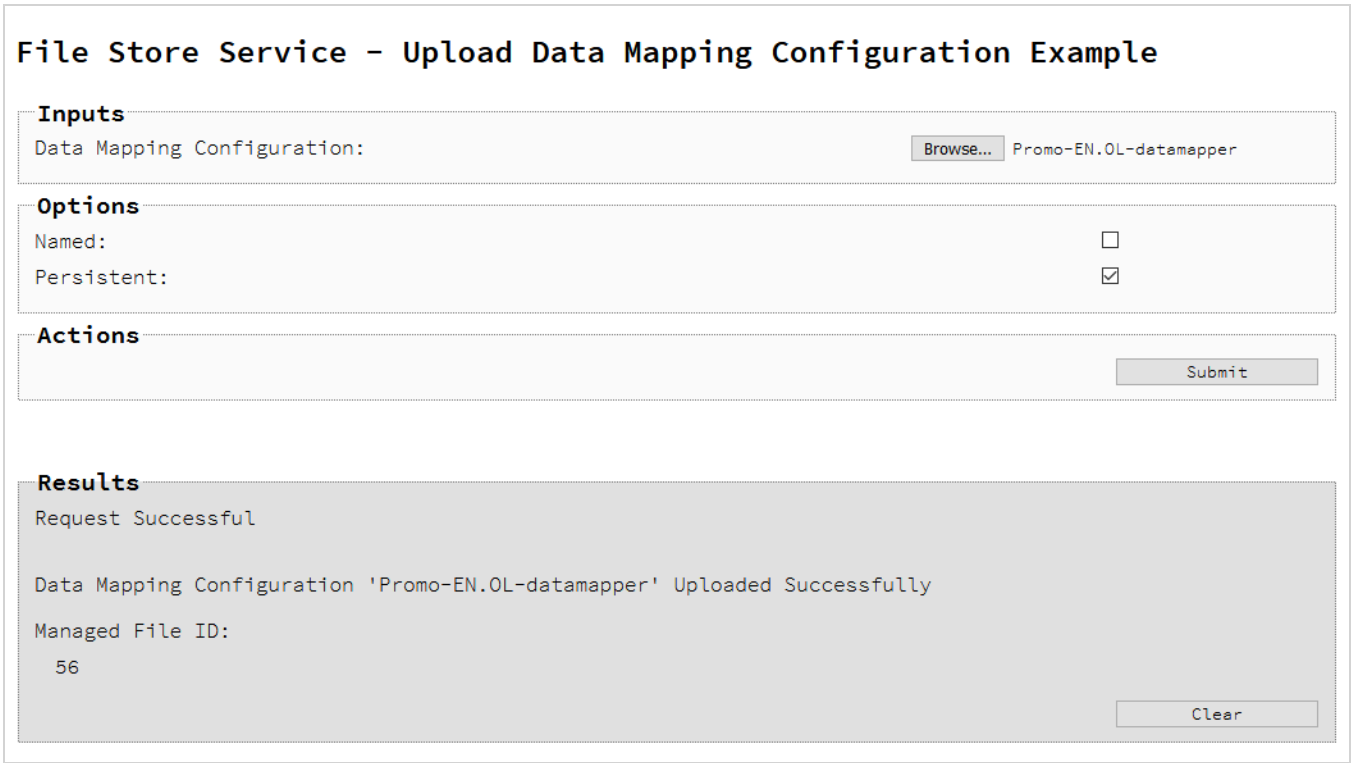
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Upload Data Mapping Configuration Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/fs-datamapper-upload.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>File Store Service - Upload Data Mapping Configuration Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datamapper">Data Mapping Configuration:</label>
                    <input id="datamapper" type="file" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="named">Named:</label>
                    <input id="named" type="checkbox">
                </div>
                <div>
                    <label for="persistent">Persistent:</label>
                    <input id="persistent" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

fs-datamapper-upload.js

```
/* File Store Service - Upload Data Mapping Configuration Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        c.setupDataMappingConfigurationFileInput($("#datamapper"));

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var file = $("#datamapper")[0].files[0],
                named = $("#named").prop("checked"),
                persistent = $("#persistent").prop("checked");

            var settings = {
                type:           "POST",
                url:            "/rest/serverengine/filestore/DataMiningConfig?persistent=" + persistent,
                data:           file,
                processData:    false,
                contentType:    "application/octet-stream"
            };
            if (named) settings.url += "&filename=" + file.name;
            $.ajax(settings)
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayInfo("Data Mapping Configuration '" + file.name + "' Uploaded Successfully");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply select the **Browse** button and then select the data mapping configuration you wish to upload using the selection dialog box.

Next you can specify the following options to use with the upload of the data mapping configuration:

- **Named** – allow this configuration to be identified/referenced by its Managed File Name as well as its Managed File ID
- **Persistent** – make this configuration persistent in the file store

> **Note:** Only one Managed File in the file store can be associated with a specific name. If two files are uploaded to the file store under the same name, then only the most recently uploaded file will be associated with (or can be referenced using) that name.

Once the configuration and options are selected, simply select the **Submit** button to upload the configuration to the server's file store and the resulting Managed File ID for the data mapping configuration will be returned and displayed to the **Results** area.

Discussion

Firstly, we define an event handler that will run in response to the submission of the HTML form via the selection of the **Submit** button.

When our event handler function is called, we then obtain a reference to the local data mapping configuration previously selected. This is achieved by getting the first value of the `files` attribute of the HTML element with the ID of `datamapper` (in this case a file type input HTML element) and storing it in a variable `file`.

We also obtain boolean values for the **Named** and **Persistent** options (both checkbox type input HTML elements) and store them in the `named` and `persistent` variables respectively.

Next we construct a jQuery AJAX request which will be sent to the File Store REST service. We use an object called `settings` to hold the arguments for our request:

Method `type` and `url` arguments are specified as shown earlier, with the addition of a `persistent` query parameter which specifies whether the configuration is to be persistent in the file store when uploaded.

We specify the variable `file` as the `data` or contents of the request, a `contentType` argument of `"application/octet-stream"`, and because we are sending file data we also specify a `processData` argument set to `false`.

If the **Named** option is checked in our form, and the `named` variable is `true`, then a `filename` query parameter is also added which contains the file name of the configuration selected (`file.name`).

Lastly, the `settings` object is passed as an argument to the jQuery AJAX function `ajax` and the request is executed.

When the request is successful or done, a request response is received and the content of that response is passed as the function parameter `response`. In the example, we then display the value of this parameter which should be the new Managed File ID of the data mapping configuration in the file store.

Further Reading

See the File Store Service page of the REST API Reference section for further detail.

Uploading a Template to the File Store

Problem

You want to upload a template to the File Store so that it can be used as part of a Content Creation operation.

Solution

The solution is to create a request using the following URI and method type to submit the template to the server via the File Store REST service:

Upload Template	/rest/serverengine/filestore/template	*POST*

Example

HTML5

fs-template-upload.html

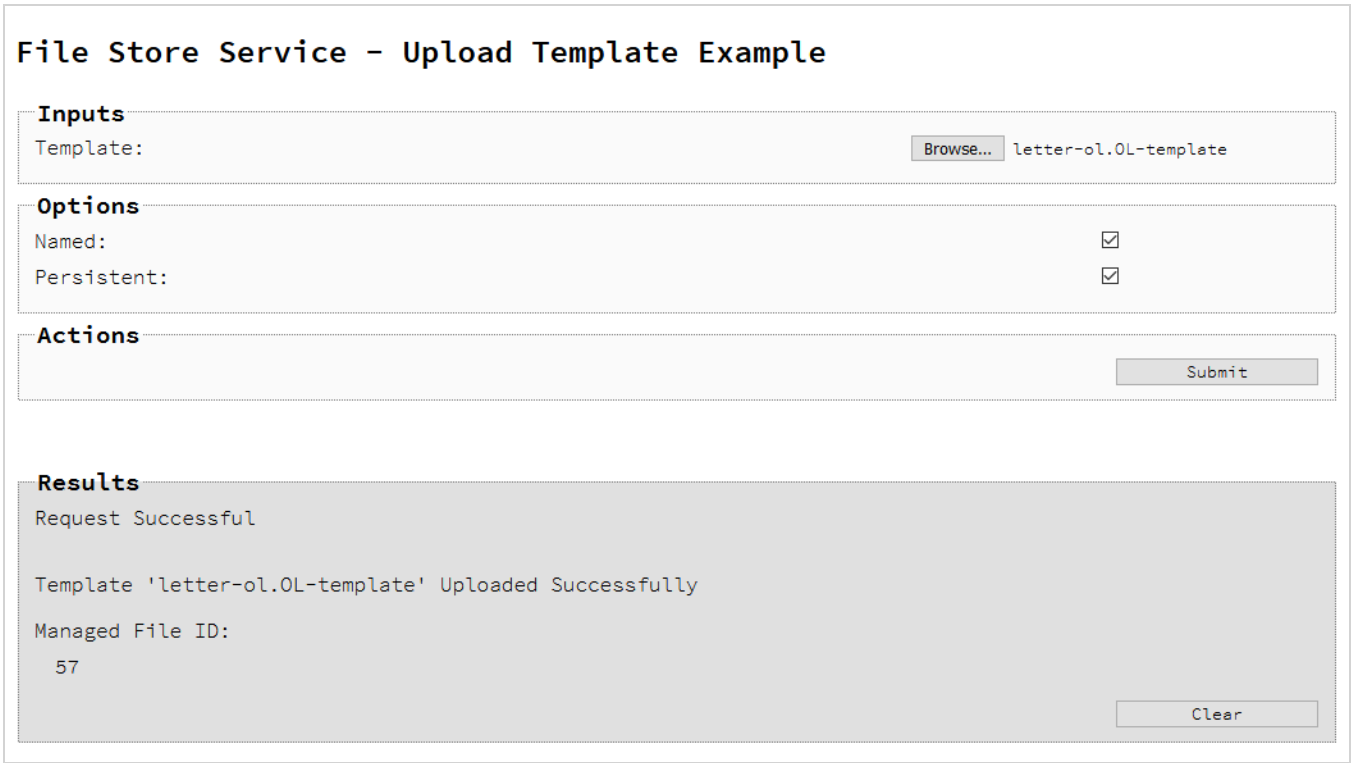
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Upload Template Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/fs-template-upload.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>File Store Service - Upload Template Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="template">Template:</label>
                    <input id="template" type="file" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="named">Named:</label>
                    <input id="named" type="checkbox" checked>
                </div>
                <div>
                    <label for="persistent">Persistent:</label>
                    <input id="persistent" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

fs-template-upload.js

```
/* File Store Service - Upload Template Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        c.setupTemplateFileInput($("#template"));

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var file = $("#template")[0].files[0],
                named = $("#named").prop("checked"),
                persistent = $("#persistent").prop("checked");

            var settings = {
                type:           "POST",
                url:            "/rest/serverengine/filestore/template?persistent=" + persistent,
                data:           file,
                processData:    false,
                contentType:    "application/zip"
            };
            if (named) settings.url += "&filename=" + file.name;
            $.ajax(settings)
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayInfo("Template '" + file.name + "' Uploaded Successfully");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply select the **Browse** button and then select the template you wish to upload using the selection dialog box.

Next you can specify the following options to use with the upload of the template:

- **Named** – allow this template to be identified/referenced by its Managed File Name as well as its Managed File ID
- **Persistent** – make this template persistent in the file store

> **Note:** Only one Managed File in the file store can be associated with a specific name. If two files are uploaded to the file store under the same name, then only the most recently uploaded file will be associated with (or can be referenced using) that name.

Once the template and options are selected, simply select the **Submit** button to upload the template to the server's file store and the resulting Managed File ID for the template will be returned and displayed to the **Results** area.

Discussion

Firstly, we define an event handler that will run in response to the submission of the HTML form via the selection of the **Submit** button.

When our event handler function is called, we then obtain a reference to the local template previously selected. This is achieved by getting the first value of the `files` attribute of the HTML element with the ID of `template` (in this case a file type input HTML element) and storing it in a variable `file`.

We also obtain boolean values for the **Named** and **Persistent** options (both checkbox type input HTML elements) and store them in the `named` and `persistent` variables respectively.

Next we construct a jQuery AJAX request which will be sent to the File Store REST service. We use an object called `settings` to hold the arguments for our request:

Method `type` and `url` arguments are specified as shown earlier, with the addition of a `persistent` query parameter which specifies whether the template is to be persistent in the file store when uploaded.

We specify the variable `file` as the `data` or contents of the request, a `contentType` argument of `"application/zip"`, and because we are sending file data we also specify a `processData` argument set to `false`.

If the **Named** option is checked in our form, and the `named` variable is `true`, then a `filename` query parameter is also added which contains the file name of the template selected (`file.name`).

Lastly, the `settings` object is passed as an argument to the jQuery AJAX function `ajax` and the request is executed.

When the request is successful or done, a request response is received and the content of that response is passed as the function parameter `response`. In the example, we then display the value of this parameter which should be the new Managed File ID of the template in the File Store.

Further Reading

See the File Store Service page of the REST API Reference section for further detail.

Uploading a Job Creation Preset to the File Store

Problem

You want to upload a job creation preset to the File Store so that it can be used as part of a Job Creation operation.

Solution

The solution is to create a request using the following URI and method type to submit the job creation preset to the server via the File Store REST service:

Upload Job Creation Preset	/rest/serverengine/filestore/JobCreationConfig	*POST*

Example

HTML5

fs-jcpreset-upload.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Upload Job Creation Preset Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/fs-jcpreset-upload.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>File Store Service - Upload Job Creation Preset Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="jcpreset">Job Creation Preset:</label>
                    <input id="jcpreset" type="file" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="named">Named:</label>
                    <input id="named" type="checkbox">
                </div>
                <div>
                    <label for="persistent">Persistent:</label>
                    <input id="persistent" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

fs-jcpreset-upload.js

```
/* File Store Service - Upload Job Creation Preset Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        c.setupJobCreationPresetFileInput($("#jcpreset"));

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var file = $("#jcpreset")[0].files[0],
                named = $("#named").prop("checked"),
                persistent = $("#persistent").prop("checked");

            var settings = {
                type:           "POST",
                url:            "/rest/serverengine/filestore/JobCreationConfig?persistent=" + persistent,
                data:           file,
                processData:    false,
                contentType:    "application/xml"
            };
            if (named) settings.url += "&filename=" + file.name;
            $.ajax(settings)
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayInfo("Job Creation Preset '" + file.name + "' Uploaded Successfully");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Usage

To run the example simply select the **Browse** button and then select the job creation preset you wish to upload using the selection dialog box.

Next you can specify the following options to use with the upload of the job creation preset:

- **Named** – allow this preset to be identified/referenced by its Managed File Name as well as its Managed File ID
- **Persistent** – make this preset persistent in the file store

> **Note:** Only one Managed File in the file store can be associated with a specific name. If two files are uploaded to the file store under the same name, then only the most recently uploaded file will be associated with (or can be referenced using) that name.

Once the preset and options are selected, simply select the **Submit** button to upload the preset to the server's file store and the resulting Managed File ID for the job creation preset will be returned and displayed to the **Results** area.

Discussion

Firstly, we define an event handler that will run in response to the submission of the HTML form via the selection of the **Submit** button.

When our event handler function is called, we then obtain a reference to the local job creation preset previously selected. This is achieved by getting the first value of the `files` attribute of the HTML element with the ID of `jcpreset` (in this case a file type input HTML element) and storing it in a variable `file`.

We also obtain boolean values for the **Named** and **Persistent** options (both checkbox type input HTML elements) and store them in the `named` and `persistent` variables respectively.

Next we construct a jQuery AJAX request which will be sent to the File Store REST service. We use an object called `settings` to hold the arguments for our request:

Method `type` and `url` arguments are specified as shown earlier, with the addition of a `persistent` query parameter which specifies whether the preset is to be persistent in the file store when uploaded.

We specify the variable `file` as the `data` or contents of the request, a `contentType` argument of `"application/xml"`, and because we are sending file data we also specify a `processData` argument set to `false`.

If the **Named** option is checked in our form, and the `named` variable is `true`, then a `filename` query parameter is also added which contains the file name of the preset selected (`file.name`).

Lastly, the `settings` object is passed as an argument to the jQuery AJAX function `ajax` and the request is executed.

When the request is successful or done, a request response is received and the content of that response is passed as the function parameter `response`. In the example, we then display the value of this parameter which should be the new Managed File ID of the job creation preset in the file store.

Further Reading

See the File Store Service page of the REST API Reference section for further detail.

Uploading an Output Creation Preset to the File Store

Problem

You want to upload an output creation preset to the File Store so that it can be used as part of a Output Creation operation.

Solution

The solution is to create a request using the following URI and method type to submit the output creation preset to the server via the File Store REST service:

Upload Output Creation Preset	/rest/serverengine/filestore/OutputCreationConfig	*POST*

Example

HTML5

fs-ocpreset-upload.html

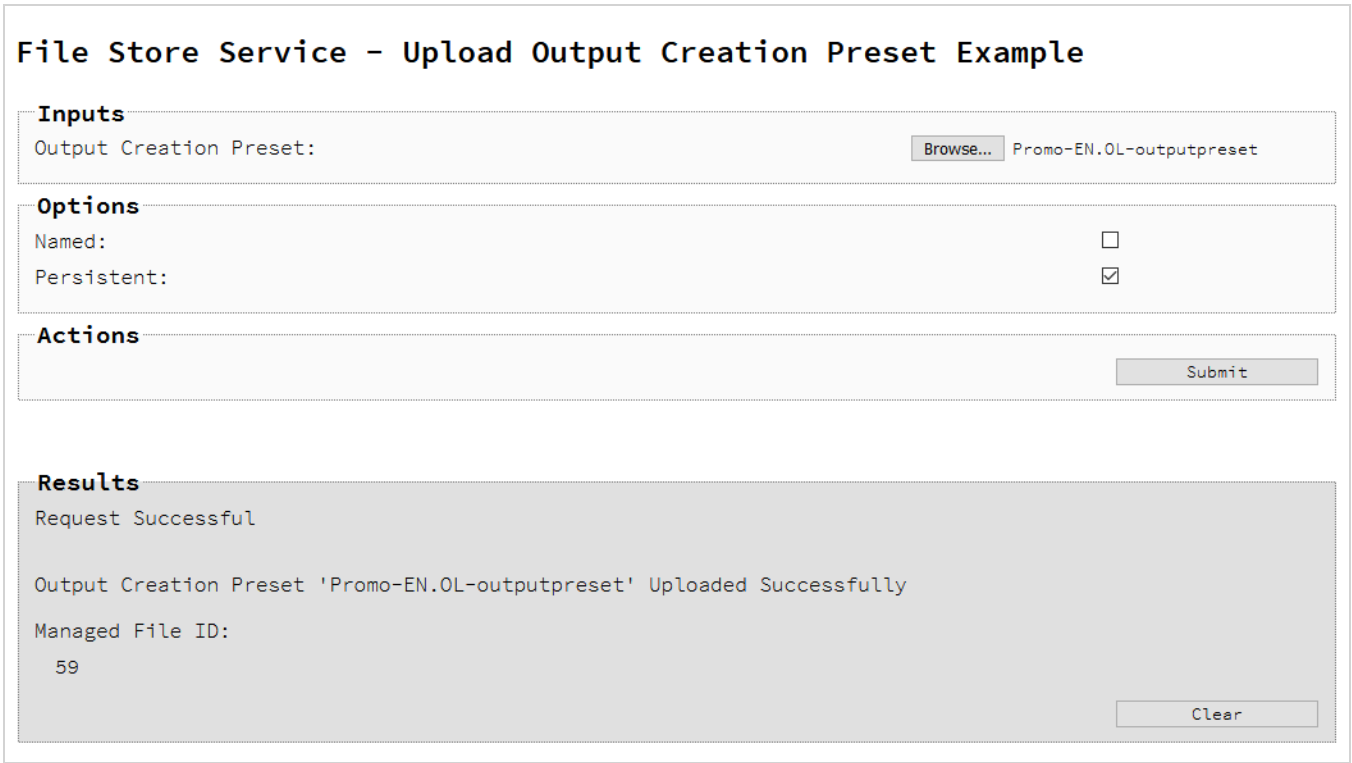
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Upload Output Creation Preset Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/fs-ocpreset-upload.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>File Store Service - Upload Output Creation Preset Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="ocpreset">Output Creation Preset:</label>
                    <input id="ocpreset" type="file" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="named">Named:</label>
                    <input id="named" type="checkbox">
                </div>
                <div>
                    <label for="persistent">Persistent:</label>
                    <input id="persistent" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

fs-ocpreset-upload.js

```
/* File Store Service - Upload Output Creation Preset Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        c.setupOutputCreationPresetFileInput($("#ocpreset"));

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var file = $("#ocpreset")[0].files[0],
                named = $("#named").prop("checked"),
                persistent = $("#persistent").prop("checked");

            var settings = {
                type:           "POST",
                url:            "/rest/serverengine/filestore/OutputCreationConfig?persistent=" + persistent,
                data:           file,
                processData:    false,
                contentType:    "application/xml"
            };
            if (named) settings.url += "&filename=" + file.name;
            $.ajax(settings)
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayInfo("Output Creation Preset '" + file.name + "' Uploaded Successfully");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply select the **Browse** button and then select the output creation preset you wish to upload using the selection dialog box.

Next you can specify the following options to use with the upload of the output creation preset:

- **Named** – allow this preset to be identified/referenced by its Managed File Name as well as its Managed File ID
- **Persistent** – make this preset persistent in the file store

> **Note:** Only one Managed File in the file store can be associated with a specific name. If two files are uploaded to the file store under the same name, then only the most recently uploaded file will be associated with (or can be referenced using) that name.

Once the preset and options are selected, simply select the **Submit** button to upload the preset to the server's file store and the resulting Managed File ID for the output creation preset will be returned and displayed to the **Results** area.

Discussion

Firstly, we define an event handler that will run in response to the submission of the HTML form via the selection of the **Submit** button.

When our event handler function is called, we then obtain a reference to the local output creation preset previously selected. This is achieved by getting the first value of the `files` attribute of the HTML element with the ID of `ocpreset` (in this case a file type input HTML element) and storing it in a variable `file`.

We also obtain boolean values for the **Named** and **Persistent** options (both checkbox type input HTML elements) and store them in the `named` and `persistent` variables respectively.

Next we construct a jQuery AJAX request which will be sent to the File Store REST service. We use an object called `settings` to hold the arguments for our request:

Method `type` and `url` arguments are specified as shown earlier, with the addition of a `persistent` query parameter which specifies whether the preset is to be persistent in the file store when uploaded.

We specify the variable `file` as the `data` or contents of the request, a `contentType` argument of `"application/xml"`, and because we are sending file data we also specify a `processData` argument set to `false`.

If the Named option is checked in our form, and the `named` variable is `true`, then a `filename` query parameter is also added which contains the file name of the preset selected (`file.name`).

Lastly, the `settings` object is passed as an argument to the jQuery AJAX function `ajax` and the request is executed.

When the request is successful or done, a request response is received and the content of that `response` is passed as the function parameter response. In the example, we then display the value of this parameter which should be the new Managed File ID of the output creation preset in the file store.

Further Reading

See the File Store Service page of the REST API Reference section for further detail.

Working with the Entity Services

This section consists of a number of pages covering various useful working examples:

1. Finding Specific Data Entities in the Server
2. Finding all the Data Sets in the Server
3. Finding the Data Records in a Data Set
4. Finding all the Content Sets in the Server
5. Finding the Content Items in a Content Set
6. Finding all the Job Sets in the Server
7. Finding the Jobs in a Job Set

See the Entity Service, Data Set Entity Service, Content Set Entity Service and Job Set Entity Service pages of the REST API Reference section for further detail.

Note: A complete listing including these examples can be found in the **index.html** file located at the root of the working example source code which contains links to all working examples.

Finding Specific Data Entities in the Server

Problem

You want to find specific Data Entities stored within the PReS Connect Server based on a set of search criteria.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Entity REST service:

Find Data Entity	/rest/serverengine/entity/find	*PUT*

Example

HTML5

e-find-data-entity.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Find Data Entity Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/e-find-data-entity.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
        <link rel="stylesheet" href="css/styles.css">
    </head>
    <body>
        <h2>Entity Service - Find Data Entity Example</h2>
        <form>
            <fieldset>
                <legend>Search Parameters</legend>
                <div>
                    <label for="entity">Entity Type:</label>
                    <select id="entity">
                        <option value="DATARECORDS">Data Records</option>
                        <option value="DATASETS">Data Sets</option>
                        <option value="CONTENTITEMS">Content Items</option>
                        <option value="CONTENTSETS">Content Sets</option>
                        <option value="JOBS">Jobs</option>
                        <option value="JOBSETS">Job Sets</option>
                    </select>
                </div>
            </fieldset>
            <fieldset id="search">
                <legend>Search Rules</legend>
                <div class="form-only">
                    <label for="rule-type">Rule Type Selector:</label>
                    <select id="rule-type">
                        <option value="NONE">No Rules</option>
                    </select>
                </div>
                <div id="RULESET" class="rule">
                    <label for="rule">Rules:</label>
                    <div id="rule" class="rule-body">
                        <div class="sub-rules">
                            <label>No Rules</label>
                        </div>
                        <div>
                            <label for="condition">Rules Condition:</label>
                            <select id="condition">
                                <option value="ALL">Match All Rules</option>
                                <option value="ANY">Match Any Rule</option>
                                <option value="NOTALL">Not Match All Rules</option>
                                <option value="NOTANY">Not Match Any Rule</option>
```

```
                        </select>
                    </div>
                    <div class="form-only">
                        <input id="add-rule" type="button" value="Add Search Rule">
                    </div>
                </div>
            </div>
        </fieldset>
        <fieldset id="sorting">
            <legend>Sorting Rules</legend>
            <div class="form-only">
                <label for="rule-type">Rule Type Selector:</label>
                <select id="rule-type">
                    <option value="NONE">No Rules</option>
                </select>
            </div>
            <div id="RULESET" class="rule">
                <label for="rule">Rules:</label>
                <div id="rule" class="rule-body">
                    <div class="sub-rules">
                        <label>No Rules</label>
                    </div>
                    <div class="form-only">
                        <input id="add-rule" type="button" value="Add Sorting Rule">
                    </div>
                </div>
            </div>
        </fieldset>
        <fieldset id="grouping">
            <legend>Grouping Rules</legend>
            <div class="form-only">
                <label for="rule-type">Rule Type Selector:</label>
                <select id="rule-type">
                    <option value="NONE">No Rules</option>
                </select>
            </div>
            <div id="RULESET" class="rule">
                <label for="rule">Rules:</label>
                <div id="rule" class="rule-body">
                    <div class="sub-rules">
                        <label>No Rules</label>
                    </div>
                    <div class="form-only">
                        <input id="add-rule" type="button" value="Add Grouping Rule">
                    </div>
                </div>
            </div>
        </fieldset>
        <fieldset>
            <legend>Actions</legend>
            <div>
                <input id="reset" type="button" value="Reset">
                <input id="submit" type="submit" value="Submit">
            </div>
        </fieldset>
    </form>
</body>
</html>
```

rules.html

```
<!-- OL Connect REST API Cookbook - Working Examples [Rules HTML Snippet] -->
<div id="search-rules" class="search">
    <div id="VALUE" class="rule entity-DATARECORDS entity-CONTENTITEMS">
        <label for="rule">Data Value Rule:</label>
        <div id="rule" class="rule-body">
            <div>
                <label for="fieldName">Field Name:</label>
                <input id="fieldName" type="text" placeholder="Field Name" required />
            </div>
            <div class="form-only">
                <label for="fieldType">Field Type:</label>
                <select id="fieldType" class="options-selector">
                    <option value="string">String</option>
                    <option value="number">Number</option>
                    <option value="boolean">Boolean</option>
                    <option value="date">Date</option>
                </select>
            </div>
```

```
            <div class="option fieldType-string compareType- condition-">
                <label for="condition">Condition:</label>
                <select id="condition" class="options-selector" data-type="enum-condition-string"></select>
            </div>
            <div class="option fieldType-number compareType- condition-">
                <label for="condition">Condition:</label>
                <select id="condition" class="options-selector" data-type="enum-condition-number"></select>
            </div>
            <div class="option fieldType-date compareType- condition-">
                <label for="condition">Condition:</label>
                <select id="condition" class="options-selector" data-type="enum-condition-date"></select>
            </div>
            <div class="option fieldType-boolean compareType- condition-">
                <label for="condition">Condition:</label>
                <select id="condition" class="options-selector" data-type="enum-condition-boolean"></select>
            </div>
            <div class="form-only option fieldType-string fieldType-number fieldType-date compareType- condition-">
                <label for="compareType">Compare Type:</label>
                <select id="compareType" class="options-selector">
                    <option value="name">Field Name</option>
                    <option value="value" selected="selected">Field Value</option>
                </select>
            </div>
            <div class="form-only option fieldType-boolean compareType- condition-EQ condition-NE">
                <label for="compareType">Compare Type:</label>
                <select id="compareType" class="options-selector">
                    <option value="name">Field Name</option>
                    <option value="value" selected="selected">Field Value</option>
                </select>
            </div>
            <div class="option fieldType-string compareType-value condition-">
                <label for="value1">Value:</label>
                <input id="value1" type="text" placeholder="Value" required />
            </div>
            <div class="option fieldType-number compareType-value condition-">
                <label for="value1">Value:</label>
                <input id="value1" type="number" step="0.001" placeholder="Value" required />
            </div>
            <div class="option fieldType-boolean compareType-value condition-EQ condition-NE">
                <label for="value1">Value:</label>
                <input id="value1" type="checkbox" />
            </div>
            <div class="option fieldType-date compareType-value condition-">
                <label for="value1">Value:</label>
                <input id="value1" type="date" required />
            </div>
            <div class="option fieldType-date compareType-value condition-BETWEEN">
                <label for="value2">Value 2:</label>
                <input id="value2" type="date" required />
            </div>
            <div class="option fieldType-string fieldType-number fieldType-date compareType-name condition-">
                <label for="value1">Field Name:</label>
                <input id="value1" type="text" placeholder="Field Name" required />
            </div>
            <div class="option fieldType-boolean compareType-name condition-EQ condition-NE">
                <label for="value1">Field Name:</label>
                <input id="value1" type="text" placeholder="Field Name" required />
            </div>
            <div class="option fieldType-date compareType-name condition-BETWEEN">
                <label for="value2">Field Name 2:</label>
                <input id="value2" type="text" placeholder="Field Name" required />
            </div>
        </div>
    </div>
    <div id="PROPERTY" class="rule entity-">
        <label for="rule">Property Value Rule:</label>
        <div id="rule" class="rule-body">
            <div>
                <label for="property">Property:</label>
                <input id="property" type="text" placeholder="Property" required />
            </div>
            <div>
                <label for="condition">Condition:</label>
                <select id="condition" data-type="enum-condition-property"></select>
            </div>
            <div class="option compareType- form-only entity-DATARECORDS entity-CONTENTITEMS">
                <label for="compareType">Compare Type:</label>
                <select id="compareType" class="options-selector">
                    <option value="name">Field Name</option>
                    <option value="value" selected="selected">Field Value</option>
                </select>
            </div>
```

```
        <div class="option compareType- form-only entity-DATASETS entity-CONTENTSETS entity-JOBS entity-JOBSETS">
            <label for="compareType">Compare Type:</label>
            <select id="compareType" class="options-selector">
                <option value="value" selected="selected">Field Value</option>
            </select>
        </div>
        <div class="option compareType-value">
            <label for="value">Value:</label>
            <input id="value" type="text" placeholder="Value" required />
        </div>
        <div class="option compareType-name entity-DATARECORDS entity-CONTENTITEMS">
            <label for="value">Field Name:</label>
            <input id="value" type="text" placeholder="Field Name" required />
        </div>
    </div>
</div>
<div id="VALUEIN" class="rule entity-">
    <label for="rule">Value In Rule:</label>
    <div id="rule" class="rule-body">
        <div>
            <label for="field">Field:</label>
            <input id="field" type="text" placeholder="Field" required />
        </div>
        <div class="option dataType- entity-DATASETS entity-CONTENTSETS entity-JOBS entity-JOBSETS">
            <label for="dataType">Data Type:</label>
            <select id="dataType" class="options-selector">
                <option value="PROPERTY" selected="selected">Property</option>
            </select>
        </div>
        <div class="option dataType- entity-DATARECORDS entity-CONTENTITEMS">
            <label for="dataType">Data Type:</label>
            <select id="dataType" class="options-selector">
                <option value="FIELD">Field</option>
                <option value="PROPERTY" selected="selected">Property</option>
            </select>
        </div>
        <div>
            <label for="condition">Condition:</label>
            <select id="condition" data-type="enum-condition-in"></select>
        </div>
        <div>
            <label for="values">Values:</label>
            <input id="values" type="text" placeholder="Value1, Value2, Value3, ..." required />
        </div>
    </div>
</div>
<div id="IDIN" class="rule entity-">
    <label for="rule">ID In Rule:</label>
    <div id="rule" class="rule-body">
        <div>
            <label for="condition">Condition:</label>
            <select id="condition" data-type="enum-condition-in"></select>
        </div>
        <div>
            <label for="values-number">Values:</label>
            <input id="values-number" type="text" placeholder="1234, 2345, 3456, ..." required />
        </div>
    </div>
</div>
<div id="DOCMEDIA" class="rule entity-CONTENTITEMS">
    <label for="rule">Document Media Rule:</label>
    <div id="rule" class="rule-body">
        <div>
            <label for="attribute">Attribute:</label>
            <select id="attribute" class="options-selector">
                <option value="NAME">Name</option>
                <option value="FRONT_COATING">Front Coating</option>
                <option value="BACK_COATING">Back Coating</option>
            </select>
        </div>
        <div>
            <label for="condition">Condition:</label>
            <select id="condition" data-type="enum-condition-contains"></select>
        </div>
        <div class="option attribute-NAME">
            <label for="name">Name:</label>
            <input id="name" type="text" placeholder="Name" required />
        </div>
        <div class="option attribute-FRONT_COATING attribute-BACK_COATING">
            <label for="coating">Coating:</label>
            <select id="coating" data-type="enum-coating"></select>
        </div>
```

```
    </div>
</div>
<div id="DOCBINDING" class="rule entity-CONTENTITEMS">
    <label for="rule">Document Binding Rule:</label>
    <div id="rule" class="rule-body">
        <div>
            <label for="attribute">Attribute:</label>
            <select id="attribute" class="options-selector">
                <option value="STYLE">Style</option>
                <option value="SIDE">Side</option>
                <option value="LOCATION">Location</option>
                <option value="ANGLE">Angle</option>
            </select>
        </div>
        <div>
            <label for="condition">Condition:</label>
            <select id="condition" data-type="enum-condition-contains"></select>
        </div>
        <div class="option attribute-STYLE">
            <label for="bindingStyle">Binding Style:</label>
            <select id="bindingStyle" data-type="enum-bindingstyle"></select>
        </div>
        <div class="option attribute-SIDE">
            <label for="bindingEdge">Binding Edge:</label>
            <select id="bindingEdge" data-type="enum-bindingedge"></select>
        </div>
        <div class="option attribute-LOCATION">
            <label for="bindingType">Binding Type:</label>
            <select id="bindingType" data-type="enum-bindingtype"></select>
        </div>
        <div class="option attribute-ANGLE">
            <label for="bindingAngle">Binding Angle:</label>
            <select id="bindingAngle" data-type="enum-bindingangle"></select>
        </div>
    </div>
</div>
<div id="DOCSIZE" class="rule entity-CONTENTITEMS">
    <label for="rule">Document Size Rule:</label>
    <div id="rule" class="rule-body">
        <div>
            <label for="entity">Entity:</label>
            <select id="entity">
                <option value="PAGE">Pages</option>
                <option value="SHEET">Sheets</option>
                <option value="SECTION">Sections</option>
            </select>
        </div>
        <div>
            <label for="condition">Condition:</label>
            <select id="condition" data-type="enum-condition-number"></select>
        </div>
        <div>
            <label for="value">Value:</label>
            <input id="value" type="number" step="1" placeholder="Value" required />
        </div>
    </div>
</div>
<div id="DUPLEX" class="rule entity-CONTENTITEMS">
    <label for="rule">Duplex Rule:</label>
    <div id="rule" class="rule-body">
        <div>
            <label for="condition">Condition:</label>
            <select id="condition" data-type="enum-condition-duplex"></select>
        </div>
    </div>
</div>
<div id="TEMPLATE" class="rule entity-CONTENTITEMS entity-CONTENTSETS">
    <label for="rule">Template Rule:</label>
    <div id="rule" class="rule-body">
        <div>
            <label for="condition">Condition:</label>
            <select id="condition" data-type="enum-condition-equal"></select>
        </div>
        <div>
            <label for="name">Name:</label>
            <input id="name" type="text" placeholder="Name" required />
        </div>
    </div>
</div>
<div id="RULESET" class="rule entity-">
    <label for="rule">Rule Set:</label>
    <div id="rule" class="rule-body">
```

```
            <div class="sub-rules">
                <label>No Rules</label>
            </div>
            <div>
                <label for="condition">Rules Condition:</label>
                <select id="condition">
                    <option value="ALL">Match All Rules</option>
                    <option value="ANY">Match Any Rule</option>
                    <option value="NOTALL">Not Match All Rules</option>
                    <option value="NOTANY">Not Match Any Rule</option>
                </select>
            </div>
        </div>
    </div>
</div>
<div id="sorting-grouping-rules" class="sorting grouping">
    <div id="value" class="rule entity-DATARECORDS entity-CONTENTITEMS">
        <label for="rule">Data Value Rule:</label>
        <div id="rule" class="rule-body">
            <div>
                <label for="name">Name:</label>
                <input id="name" type="text" placeholder="Name" required />
            </div>
            <div>
                <label for="numeric">Numeric:</label>
                <input id="numeric" type="checkbox" />
            </div>
            <div>
                <label for="order">Order:</label>
                <select id="order">
                    <option value="ASC">Ascending</option>
                    <option value="DESC">Descending</option>
                </select>
            </div>
        </div>
    </div>
    <div id="property" class="rule entity-">
        <label for="rule">Property Value Rule:</label>
        <div id="rule" class="rule-body">
            <div>
                <label for="name">Name:</label>
                <input id="name" type="text" placeholder="Name" required />
            </div>
            <div>
                <label for="order">Order:</label>
                <select id="order">
                    <option value="ASC">Ascending</option>
                    <option value="DESC">Descending</option>
                </select>
            </div>
        </div>
    </div>
</div>
<div id="rule-data-types">
    <select id="enum-condition-string-options">
        <option value="EQ" selected="selected">=</option>
        <option value="NE">!=</option>
        <option value="CONTAINS">Contains</option>
        <option value="NOT_CONTAINS">Not Contains</option>
        <option value="STARTS_WITH">Starts with</option>
        <option value="ENDS_WITH">Ends with</option>
        <option value="LIKE">Like</option>
        <option value="NOT_LIKE">Not Like</option>
    </select>
    <select id="enum-condition-number-options">
        <option value="EQ" selected="selected">=</option>
        <option value="NE">!=</option>
        <option value="LT">&lt;</option>
        <option value="GT">&gt;</option>
        <option value="LTE">&lt;=</option>
        <option value="GTE">&gt;=</option>
    </select>
    <select id="enum-condition-date-options">
        <option value="EQ" selected="selected">=</option>
        <option value="NE">!=</option>
        <option value="LT">&lt;</option>
        <option value="GT">&gt;</option>
        <option value="LTE">&lt;=</option>
        <option value="GTE">&gt;=</option>
        <option value="BETWEEN">Between</option>
    </select>
    <select id="enum-condition-boolean-options">
```

```
        <option value="EQ" selected="selected">=</option>
        <option value="NE">!=</option>
        <option value="IS_TRUE">Is True</option>
        <option value="IS_FALSE">Is False</option>
    </select>
    <select id="enum-condition-property-options">
        <option value="EQ" selected="selected">=</option>
        <option value="NE">!=</option>
        <option value="LT">&lt;</option>
        <option value="GT">&gt;</option>
        <option value="LTE">&lt;=</option>
        <option value="GTE">&gt;=</option>
        <option value="CONTAINS">Contains</option>
        <option value="NOT_CONTAINS">Not Contains</option>
        <option value="STARTS_WITH">Starts with</option>
        <option value="ENDS_WITH">Ends with</option>
        <option value="LIKE">Like</option>
        <option value="NOT_LIKE">Not Like</option>
    </select>
    <select id="enum-condition-equal-options">
        <option value="EQ" selected="selected">=</option>
        <option value="NE">!=</option>
    </select>
    <select id="enum-condition-contains-options">
        <option value="CONTAINS" selected="selected">Contains</option>
        <option value="NOT_CONTAINS">Not Contains</option>
    </select>
    <select id="enum-condition-in-options">
        <option value="IN" selected="selected">In</option>
        <option value="NOT_IN">Not In</option>
    </select>
    <select id="enum-condition-duplex-options">
        <option value="SIMPLEX_ONLY" selected="selected">Simplex Only</option>
        <option value="HAS_DUPLEX">Has Duplex</option>
    </select>
    <select id="enum-coating-options">
        <option value="UNSPECIFIED" >Unspecified</option>
        <option value="NONE" selected="selected">None</option>
        <option value="COATED">Coated</option>
        <option value="GLOSSY">Glossy</option>
        <option value="HIGH_GLOSS">High Gloss</option>
        <option value="INKJET">Inkjet</option>
        <option value="MATTE">Matte</option>
        <option value="SATIN">Satin</option>
        <option value="SEMI_GLOSS">Semi Gloss</option>
    </select>
    <select id="enum-bindingstyle-options">
        <option value="NONE">None</option>
        <option value="DEFAULT" selected="selected">Default</option>
        <option value="STAPLED">Stapled</option>
        <option value="GLUED">Glued</option>
        <option value="STITCHED">Stitched</option>
        <option value="ADHESIVE">Adhesive</option>
        <option value="SPINETAPING">Spine Taping</option>
        <option value="RING">Ring</option>
        <option value="WIREDCOMB">Wired Comb</option>
        <option value="PLASTICCOMB">Plastic Comb</option>
        <option value="COIL">Coil</option>
    </select>
    <select id="enum-bindingedge-options">
        <option value="DEFAULT" selected="selected">Default</option>
        <option value="LEFT">Left</option>
        <option value="RIGHT">Right</option>
        <option value="TOP">Top</option>
        <option value="BOTTOM">Bottom</option>
    </select>
    <select id="enum-bindingtype-options">
        <option value="DEFAULT" selected="selected">Default</option>
        <option value="SADDLE">Saddle</option>
        <option value="SIDE">Side</option>
        <option value="CORNER">Corner</option>
    </select>
    <select id="enum-bindingangle-options">
        <option value="DEFAULT" selected="selected">Default</option>
        <option value="VERTICAL">Vertical</option>
        <option value="HORIZONTAL">Horizontal</option>
        <option value="ANGLE">Angle</option>
    </select>
</div>
```

JavaScript/jQuery

e-find-data-entity.js

```
/* Entity Service - Find Data Entity Example */
(function ($, c) {
    "use strict";
    $(function () {

        const
            MSG_LOAD_RULE_FAIL = "Loading of Search Rules Unsuccessful!\n\n" +
            "Unable to load the search rules from the search rules template. " +
            "Searching is currently disabled.",

            MSG_INCOMPAT_RULES = "The entity type selected isn't compatible " +
            "with some of the existing rules and these will need to be " +
            "removed.\n\nAre you sure you wish to continue ?",

            MSG_MULTIPLE_RULES = "The rule set rule being removed contains " +
            "multiple rules.\n\nAre you sure you wish to continue ?",

            MSG_RESET_RULES = "Are you sure you wish to continue ?";

        c.setupExample();

        var $allRules;

        /* Load Rules */
        (function () {
            var $temp = $("<div>");
            $temp.load("snippets/rules.html", function (response, status) {
                var success = (status === "success");
                if (!success)
                    alert(MSG_LOAD_RULE_FAIL);
                else {
                    $allRules = $temp;
                    ["search", "sorting", "grouping"].forEach(function (category) {
                        var $selector = $("fieldset#" + category)
                                        .find("#rule-type")
                                        .empty();
                        $allRules
                            .find("div." + category + " div.rule")
                            .each(function (index, rule) {
                                var label = $(rule)
                                                .children("label").text()
                                                .replace(/(\sRule)?\:$/, '');
                                $selector.append($("<option>")
                                                .attr("value", rule.id)
                                                .attr("class", $(rule).attr("class"))
                                                .text(label));
                            });
                    });
                    $("#entity").trigger("change");
                }
                $("input, select").prop("disabled", !success);
            });
        }());

        /* Common Load Rule Function */
        function loadRule(category, ruleType) {
            var $rule = $allRules
                            .children("div." + category)
                            .find("div.rule[id='" + ruleType + "']")
                            .clone();

            /* Populate any Data Type References */
            $rule.find("select[data-type]").each(function (index, element) {
                var $element = $(element),
                    dataType = $element.attr("data-type");
                var options = $allRules
                                .find("#rule-data-types")
                                .find("#" + dataType + "-options")
                                .children();
                $element
                    .empty()
                    .append(options.clone());
                $element.val($element
                                .find("option[selected]")
```

```
                        .val());
    });

    /* Allow Rules to be Draggable by Label */
    $rule.children("label")
        .prop("draggable", true)
        .addClass("draggable");

    /* Append Add / Remove Rule Buttons */
    var $buttons = $("<div>", { "class": "form-only" }).append(
            $("<input>", { "id": "remove-rule", "type": "button",
                "value": "Remove Rule" }));
    if (ruleType === "RULESET") {
        $buttons
            .append(
                $("<input>", { "id": "add-rule", "type": "button",
                    "value": "Add Search Rule" }))
            .children("#remove-rule")
            .attr("value", "Remove Set");
    }
    $rule.children("div.rule-body").append($buttons);
    return $rule;
}

/* Manage the Available Rule Types based on Entity Type */
$("#entity")
    .on("click", function (event) {
        var $entity = $(event.target);
        $entity.data("previous", $entity.val());
    })
    .on("change", function (event) {
        var $entity = $(event.target),
            categories = ["search", "sorting", "grouping"],
            options = {},
            incompatible = {},
            reconfigure = {};

        categories.forEach(function (category) {
            options[category] = [];
            $("fieldset#" + category)
                .find("#rule-type")
                .children("option")
                .each(function (index, option) {
                    var $option = $(option),
                        allClazz = $entity.attr("id") + "-";

                    if ($option.hasClass(allClazz) ||
                        $option.hasClass(allClazz + $entity.val()))
                        options[category].push($option.val());
                });
        });

        /**
         * Prompt User & Remove any Existing Rules that are
         * incompatible with currently Entity type selected
         */
        categories.forEach(function (category) {
            $("fieldset#" + category)
                .children("div#RULESET")
                .find("div.rule")
                .each(function (index, rule) {
                    if ($.inArray(rule.id, options[category]) < 0) {
                        if (incompatible[category] === undefined)
                            incompatible[category] = [];
                        incompatible[category].push(index);
                    } else
                        $entity
                            .children("option")
                            .toArray()
                            .map(function (option) {
                                return $(option).val();
                            }).forEach(function (entity) {
                                var type = $entity.attr("id") + "-" + entity;
                                if ($(rule).find("div.option").hasClass(type)) {
                                    if (reconfigure[category] === undefined)
                                        reconfigure[category] = [];
                                    reconfigure[category].push(index);
                                }
                            });
                });
        });
        if (Object.keys(incompatible).length > 0 &&
```

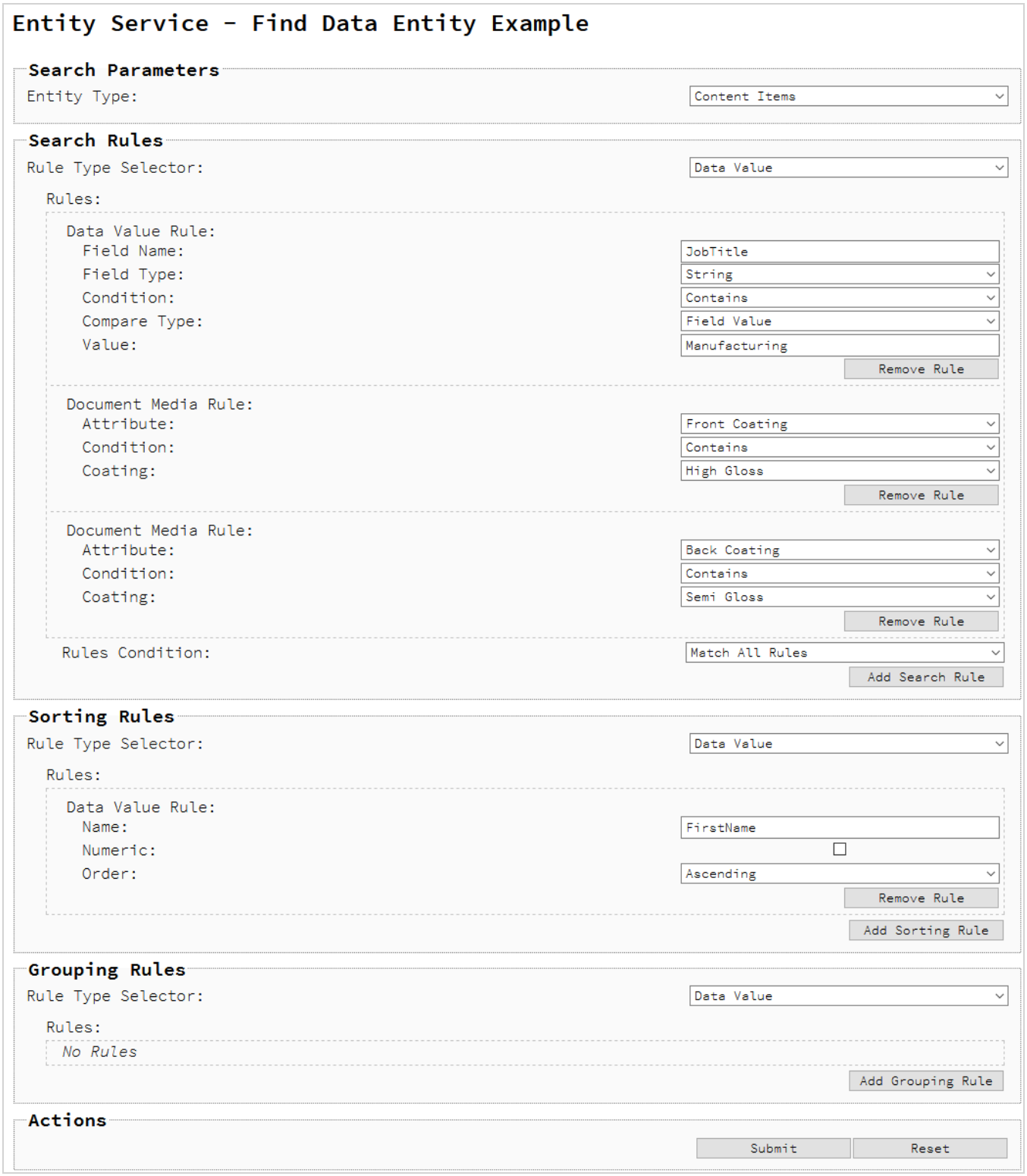
```
            !confirm(MSG_INCOMPAT_RULES)) {
            $entity.val($entity.data("previous"));
            return;
        }
        categories.forEach(function (category) {
            var $rules = $("fieldset#" + category)
                            .children("div#RULESET")
                            .find("div.rule");

            /* Remove any incompatible rules */
            if (incompatible[category] !== undefined)
                for (var i = 0; i < incompatible[category].length; i += 1)
                    $($rules[incompatible[category][i]]).remove();

            /* Reconfigure any incompatible options */
            if (reconfigure[category] !== undefined)
                for (var j = 0; j < reconfigure[category].length; j += 1)
                    $($rules[reconfigure[category][j]])
                        .find("div.option")
                        .children("select.options-selector")
                        .trigger("change");

            /* Restrict Rule Type Selectors to Entity Specific Options */
            var $selector = $("fieldset#" + category).find("#rule-type"),
                selection = $selector.val();
            $selector
                .children()
                .each(function (index, option) {
                    var $option = $(option),
                        name = $option.val(),
                        invalid = ($.inArray(name, options[category]) < 0);
                    if (invalid && selection === name)
                        selection = null;
                    $option
                        .prop("disabled", invalid)
                        .prop("hidden", invalid);
                })
                .each(function (index, option) {
                    var $option = $(option);
                    if (selection === null && !$option.prop("disabled")) {
                        $selector.val($option.val());
                        return false;
                    }
                });
        });
    });

/* Process Rules Function */
function processRules($rules) {
    var rules = [],
        typeKey = "ruleType";
    if ($rules.closest("fieldset").attr("id") !== "search")
        typeKey = "type";

    $rules
        .children("div.rule")
        .each(function (index, element) {
            var ruleType = element.id,
                fieldType = null,
                $body = $(element).children("div.rule-body"),
                rule = {};

            if ($body.find("div:visible #compareType").val() === "name")
                fieldType = "FIELD";
            else if ($body.find("div #fieldType").val() === "date")
                fieldType = "DATE";

            $body
                .children("div")
                .not(".form-only")
                .children(":visible :input")
                .each(function (index, input) {
                        var $input = $(input),
                            id = $input.attr("id"),
                            type = $input.attr("type"),
                            value;

                        if (id !== undefined && !$input.prop("disabled")) {
                            if (id.match(/^value(s|\d*)\-/) !== null) {
                                type = id.split("-")[1];
                                id = id.split("-")[0];
                            }
```

```
                            if (id === "values") {
                                value = $input.val().split(/\s*,\s*/);
                                if (type === "number")
                                    for (var i = 0; i < value.length; i += 1)
                                        value[i] = c.valueToNumber(value[i]);
                            }
                            if (!Array.isArray(value)) {
                                if (type === "checkbox")
                                    value = $input.prop("checked");
                                else if (type === "number")
                                    value = c.valueToNumber($input.val());
                                else
                                    value = $input.val();
                            }

                            if (fieldType !== null && id.match(/^value\d*$/))
                                value = {
                                    "type": fieldType,
                                    "value": value
                                };

                            rule[id] = value;
                        }
                    });

                if (ruleType === "RULESET")
                    rule.rules = processRules($body.children("div.sub-rules"));

                rules.push($.extend(true, { [typeKey]: ruleType }, rule));
            });
        return rules;
    }

    $("form")

        /* Add Rule Handler */
        .on("click", "input#add-rule", function (event) {
            var $parent = $(event.target).closest("fieldset"),
                $rule = loadRule($parent.attr("id"), $parent
                                                        .find("#rule-type")
                                                        .val());
            $(event.target)
                .closest(".rule")
                .children("div.rule-body")
                .children("div.sub-rules")
                .append($rule);
            $rule
                .find("div.rule-body div")
                .children("select.options-selector:visible")
                .trigger("change");
        })

        /* Remove Rule Handler */
        .on("click", "input#remove-rule", function (event) {
            var $rule = $(event.target).closest(".rule"),
                remove = true;
            if ($rule.attr("id") === "RULESET" &&
                    $rule.find("div.sub-rules div.rule").length > 1)
                if (!confirm(MSG_MULTIPLE_RULES))
                    remove = false;

            if (remove) $rule.remove();
        })

        /* "Select Rule Options" Change Handler */
        .on("change", "select.options-selector", function (event) {

            var $entity = $("#entity"),
                types = $entity
                            .children("option")
                            .toArray()
                            .map(function (option) {
                                return $(option).val();
                            }),
                $rule = $(event.target).closest(".rule"),
                $selectors = $rule
                                .find("div.rule-body div")
                                .children("select.options-selector:visible"),
                unhidden = [];

            $rule
```

```
        .find("div.rule-body div.option")
        .each(function (optionIndex, option) {
            var selected = true,
                $option = $(option),
                before = $option.prop("hidden"),
                matched = [];

            types.forEach(function (type) {
                var clazz = $entity.attr("id") + "-" + type;
                if ($option.hasClass(clazz))
                    matched.push(clazz);
            });
            if (matched.length)
                selected = ($.inArray($entity.attr("id") + "-" +
                    $entity.val(), matched) >= 0);

            $selectors.each(function (selectorIndex, selector) {
                var allClass = selector.id + "-";
                if (!$option.hasClass(allClass) &&
                    !$option.hasClass(allClass + $(selector).val())) {
                    selected = false;
                }
            });

            $option
                .prop("hidden", !selected)
                .find("input, select")
                .prop("disabled", !selected);

            if (before && selected)
                unhidden.push($option
                    .children("select.options-selector"));
    });

    unhidden.forEach(function (selector) {
        $(selector).trigger("change");
    });
})

/* Submit Form Rules Handler */
.on("submit", function (event) {

    event.preventDefault();
    if (!c.checkSessionValid()) return;

    /* Process & Add Search Rules */
    var search = {
        "entity": $("#entity").val(),
        "search": processRules(($("#search")))[0],
    };

    /* Process & Add Sorting & Grouping Rules */
    ["sort", "group"].forEach(function (type) {
        var rules = (processRules($("#" + type +"ing")))[0].rules;
        if (rules.length)
            search[type] = rules;
    });

    $.ajax({
        type:           "PUT",
        url:            "/rest/serverengine/entity/find",
        data:           JSON.stringify(search),
        contentType:    "application/json"
    })
        .done(function (response) {
            c.displayStatus("Request Successful");
            c.displayHeading("Input Parameters");
            c.displaySubResult("JSON Search Parameters", c.jsonPrettyPrint(search));
            c.displayHeading("Search Results");
            c.displaySubResult("Plain", c.jsonIDListsWithSortKeyToTable(response));
            c.displaySubResult("JSON Identifier Lists (with Sort Key)", c.jsonPrettyPrint(response));
        })
        .fail(c.displayDefaultFailure);
})

/* Reset Form Rules Handler */
.on("click", "#reset", function (event) {
    if (confirm(MSG_RESET_RULES))
        $("div.sub-rules")
            .find("div.rule")
            .remove();
});
```

```
    });
}(jQuery, Common));
```

Screenshot & Output



Results

Request Successful

Input Parameters:

JSON Search Parameters:


```
{
    "entity": "CONTENTITEMS",
    "search": {
        "ruleType": "RULESET",
        "condition": "ALL",
        "rules": [
            {
                "ruleType": "VALUE",
                "fieldName": "JobTitle",
                "condition": "CONTAINS",
                "value1": "Manufacturing"
            },
            {
                "ruleType": "DOCMEDIA",
                "attribute": "FRONT_COATING",
                "condition": "CONTAINS",
                "coating": "HIGH_GLOSS"
            },
            {
                "ruleType": "DOCMEDIA",
                "attribute": "BACK_COATING",
                "condition": "CONTAINS",
                "coating": "SEMI_GLOSS"
            }
        ]
    },
    "sort": [
        {
            "type": "value",
            "name": "FirstName",
            "numeric": false,
            "order": "ASC"
        }
    ]
}
```


Search Results:

Plain:

Group	Data Entity ID	Sort Key
Group 1	7510	Brandy
	7502	Dianne

JSON Identifier Lists (with Sort Key):


```
[
    [
        {
            "identifier": 7510,
            "sortkey": "Brandy"
        },
        {
            "identifier": 7502,
            "sortkey": "Dianne"
        }
    ]
]
```


Clear

Usage

To run the example first select the **Entity Type** that you are searching for. The data entity types available are:

- Data Sets
- Data Records
- Content Sets
- Content Items
- Job Sets
- Jobs

Once a data entity type is selected, various rules can be added to form the search criteria. There are three categories of rules available: *search*, *sorting* and *grouping* rules.

Search Rules

There are eight types of search rules that can be specified as part of the overall search criteria:

- **Data Value** – Search for data entities based on the value of a data record field.
- **Property Value** – Search for data entities based on the value of a data entity property.
- **Value In** – Restrict the search to the data entity values contained within a list.
- **ID In** – Restrict the search to data entity identifiers contained within a list.
- **Document Media** – The name of the media used as defined in the PReS Connect template, as well as the coating used for the front and back of the page sheet.
- **Document Binding** – The binding used for the document including style, edge, type and angle properties.
- **Document Size** – The document size. This supports sheet and section size counts in addition to page size counts.
- **Duplex** – Whether the document contains any duplex sheets, or not.
- **Template** – Search for data entities based on the name of the template used during Content Creation.
- **Rule Set** – Used to group search rules into logical sets (or sub-sets) and specifies a group rules operator that can be configured to either match all or any of the rules in the set. This allows quite complex nested rules.

The types of search rules available are specific to the data entity type selected. The following table lists the available combinations:

Rule Type	Data Records	Data Sets	Content Items	Content Sets	Jobs	Job Sets
Data Value	✓		✓			
Property Value	✓	✓ ^	✓	✓ ^	✓ ^	✓ ^
Value In	✓	✓ ^	✓	✓ ^	✓ ^	✓ ^
ID In	✓	✓	✓	✓	✓	✓
Document Media			✓			
Document Binding			✓			
Document Size			✓			
Duplex			✓			
Template			✓	✓		
Rule Set	✓	✓	✓	✓	✓	✓

^ **Note**: These rules types are only partially compatible with these entity data types. Only searches specific to *property values* are permitted.

Search rules can be added by selecting the appropriate **Rule Type Selector** option and then clicking the **Add Search Rule** button. They can be removed using the **Remove Rule** button, and even re-ordered within the form by dragging and dropping a rule by their name label (e.g. *Data Value Rule:* or *Property Value Rule:*).

Data Value search rules can be configured by specifying the following options:

- **Name** – Name of the data record field to search by
- **Condition** – Operator for the comparison of the data record field (e.g. *Equals (=)*, *Not Equals (<>)*, *Less Than (<)*, *Greater Than (>)*, etc.)
- **Value** – Value of the data record field to match

Property Value search rules can be configured by specifying the following options:

- **Name** – Name of the data entity property to search by
- **Condition** – Operator for the comparison of the data entity property (e.g. *Equals (=)*, *Not Equals (<>)*, *Less Than (<)*, *Greater Than (>)*, etc.)
- **Value** – Value of the data entity property to match

Value In and *ID In* search rules can be configured by specifying the following options:

- **Identifiers** – List of data entity identifiers to match or not match against

Document Media search rules for *Media Name* can be further configured by specifying the following options:

- **Condition** – Operator for the comparison of the media name (e.g. *Equals (=)* or *Not Equals (<>)*)
- **Value** – Value of the media name to match (e.g. *Plain Letter Paper*)

Document Media search rules for *Coating* can be further configured by specifying the following options:

- **Condition** – Operator for the comparison of the coating (e.g. *Equals (=)* or *Not Equals (<>)*)
- **Front Coating** – The type of front coating to match (e.g. *Semi Gloss*, *Satin*, *Matte*, *Glossy*, *None*, etc.)
- **Back Coating** – The type of back coating to match (e.g. *Semi Gloss*, *Satin*, *Matte*, *Glossy*, *None*, etc.)

Document Binding search rules can be further configured by specifying the following options:

- **Binding Style** – The style of binding to match (e.g. *Stapled*, *Glued*, *Stitched*, *Coil*, etc.)
- **Binding Edge** – The edge (or side on which the binding occurs) to match (e.g. *Left*, *Right*, *Top* or *Bottom*)
- **Binding Type** – The type or location of the binding to match (e.g. *Saddle*, *Side* or *Corner*)
- **Binding Angle** – The binding angle to match (e.g. *Vertical*, *Horizontal* or *Angle*)

Document Size search rules can be configured by specifying the following options:

- **Condition** – Operator for the comparison of the document length (e.g. *Equals (=)*, *Not Equals (<>)*, *Less Than (<)*, *Greater Than (>)*, etc.)
- **Value** – Value of the document length to match

Template search rules can be configured by specifying the following options:

- **Condition** – Operator for the comparison of the template name (e.g. *Equals (=)* or *Not Equals (<>)*)
- **Value** – Value of the template name to match (e.g. *letter-ol*)

Rule Set can be configured by specifying the following conditions:

- **Rules Conditions** – Conditions for the matching of search rules contained in the rule set. The options are *Match All Rules (ALL)*; *Match Any Rule (ANY)*; *Not Match All Rules (NOTALL)*; *Not Match Any Rule (NOTANY)*

Individual rules can be added to a *Rule Set* by selecting the appropriate **Rule Type Selector** option and then clicking the **Add Search Rule** button within the *Rule Set* box.

Individual rules can be removed by clicking the associated **Remove Rule** button within the *Rule Set* box.

Rule Sets can be removed using the **Remove Set** button, and in situations where removing a rule set would remove *multiple* rules, you will be prompted to confirm the removal of the rule set.

Lastly, select the **Rules Operator** for the matching of search rules contained in the base rules list (e.g. *Match All Rules (AND)* or *Match Any Rules (OR)*)

Sorting Rules

There are also two types of sorting rules that can be used as part of the overall search criteria:

- **Data Value** – Sort the search results by the value of a data record field
- **Property Value** – Sort the search results by the value of a data entity property

The types of sorting rules available are also specific to the data entity type selected. The following table lists the available combinations:

Rule Type	Data Records	Data Sets	Content Items	Content Sets	Jobs	Job Sets
Data Value	✓		✓			
Property Value	✓	✓	✓	✓	✓	✓

Sorting rules can be added using the **Add Sorting Rule** button, removed using the **Remove Rule** button, and even re-ordered within the form.

Rules can be re-ordered by dragging and dropping a rule by it's name label (e.g. *Data Value Rule:* or *Property Value Rule:*).

Data Value sorting rules can be configured by specifying the following options:

- **Name** – Name of the data record field to sort the search results by
- **Numeric** – Sort the search results for this data record field numerically
- **Order** – Sort the search results for this data record field in a specific order (e.g. *Ascending* or *Descending*)

Property Value sorting rules can be configured by specifying the following options:

- **Name** – Name of the data entity property to sort the search results by
- **Order** – Sort the search results for this data entity property in a specific order (e.g. *Ascending* or *Descending*)

Grouping Rules

There are also two types of grouping rules that can be used as part of the overall search criteria:

- **Data Value** – Group the search results by the value of a data record field
- **Property Value** – Group the search results by the value of a data entity property

The types of grouping rules available are also specific to the data entity type selected. The following table lists the available combinations:

Rule Type	Data Records	Data Sets	Content Items	Content Sets	Jobs	Job Sets
Data Value	✓		✓			
Property Value	✓	✓	✓	✓	✓	✓

Grouping rules can be added using the **Add Grouping Rule** button, removed using the **Remove Rule** button, and even re-ordered within the form.

Rules can be re-ordered by dragging and dropping a rule by it's name label (e.g. *Data Value Rule:* or *Property Value Rule:*).

Data Value grouping rules can be configured by specifying the following options:

- **Name** – Name of the data record field to group the search results by
- **Numeric** – Group the search results for this data record field numerically
- **Order** – Group the search results for this data record field in a specific order (e.g. *Ascending* or *Descending*)

Property Value grouping rules can be configured by specifying the following options:

- **Name** – Name of the data entity property to group the search results by
- **Order** – Group the search results for this data entity property in a specific order (e.g. *Ascending* or *Descending*)

Note: By default, comparison conditions in *search* rules are evaluated ***alphanumerically***, regardless of the type of value.

Numeric evaluation of comparison conditions is **not currently supported** in the PReS Connect REST API.

The only exception to this rule is the ability to numerically sort or group results by specifying *sorting* or *grouping* rules of a *Data Value* type.

Caution: The **Entity Type** selected for the search criteria can be changed during or even after rules have been added. But because certain rules are only available for certain data entity types, some of the existing rules in the search criteria may become ***incompatible***.
In situations where incompatible rules are found in the existing search criteria, you will be prompted to confirm the change of entity type. If you then proceed with the change of entity type, any incompatible rules found in the existing search criteria will be ***removed***.

Once the search criteria is constructed, and the required inputs populated, simply select the **Submit** button. This will submit the request to the server and display the search criteria specified as input to the **Results** area in JSON Search Parameters format.

The result will then be returned as a list of Data Entity IDs which will be appended to the **Results** area in both Plain table and JSON Identifier Lists (with Sort Key) formats.

To construct a new search criteria, the **Reset** button can be selected. This will reset the form, removing ***all*** existing rules.

Further Reading

See the Entity Service page of the REST API Reference section for further detail.

Finding all the Data Sets in the Server

Problem

You want to obtain a list of all the previously created Data Sets contained in the PReS Connect Server potentially for use in a Content Creation operation.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Data Set Entity REST service:

Get All Data Sets	/rest/serverengine/entity/datasets	*GET*

Example

HTML5

dse-get-all-datasets.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get All Data Sets Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dse-get-all-datasets.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Data Set Entity Service - Get All Data Sets Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="submit">No Input Required</label>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

dse-get-all-datasets.js

```
/* Data Set Entity Service - Get All Data Sets Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            $.ajax({
                type:   "GET",
```

```
                url:    "/rest/serverengine/entity/datasets"
            })
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayHeading("Data Set IDs");
                    c.displaySubResult("Plain", c.jsonIDListToPlain(response));
                    c.displaySubResult("JSON Identifier List", c.jsonPrettyPrint(response));
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Usage

To run the example simply select the **Submit** button to request a list of the all the data sets currently contained within the server.

The resulting list will then be returned and displayed to the **Results** area in both Plain list and JSON Identifier List formats.

Further Reading

See the Data Set Entity Service page of the REST API Reference section for further detail.

Finding the Data Records in a Data Set

Problem

You want to obtain a list of all the previously created Data Records contained within a specific Data Set potentially for use in a Content Creation operation.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Data Set Entity REST service:

Get Data Records for Data Set	/rest/serverengine/entity/datasets/{dataSetId}	*GET*

Example

HTML5

dse-get-datarecords.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get Data Records for Data Set Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dse-get-datarecords.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Data Set Entity Service - Get Data Records for Data Set Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="dataset">Data Set ID:</label>
                    <input id="dataset" type="text" placeholder="1234" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

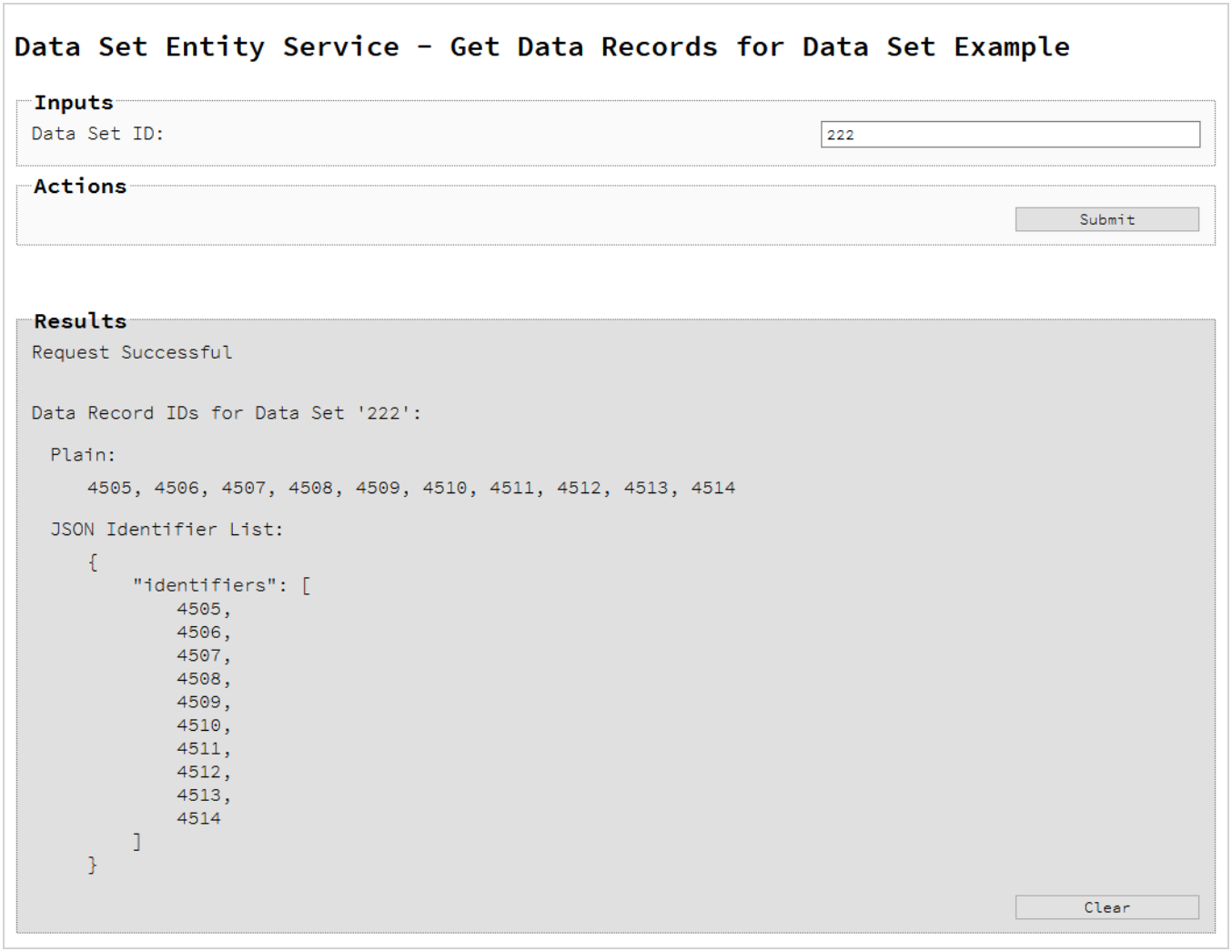
JavaScript/jQuery

dse-get-datarecords.js

```
/* Data Set Entity Service - Get Data Records for Data Set Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {
```

```
            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataSetId = $("#dataset").val();

            $.ajax({
                type:   "GET",
                url:    "/rest/serverengine/entity/datasets/" + dataSetId
            })
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayHeading("Data Record IDs for Data Set '" + dataSetId + "'");
                    c.displaySubResult("Plain", c.jsonIDListToPlain(response));
                    c.displaySubResult("JSON Identifier List", c.jsonPrettyPrint(response));
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Data Set ID** and select the **Submit** button to request a list of the all the data records contained within the specific data set in the server.

The resulting list will then be returned and displayed to the **Results** area in both Plain list and JSON Identifier List formats.

Further Reading

See the Data Set Entity Service page of the REST API Reference section for further detail.

Finding all the Content Sets in the Server

Problem

You want to obtain a list of all the previously created Content Sets contained in the PReS Connect Server potentially for use in a Job Creation operation.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Set Entity REST service:

Get All Content Sets	/rest/serverengine/entity/contentsets	*GET*

Example

HTML5

cse-get-all-contentsets.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get All Content Sets Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cse-get-all-contentsets.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Set Entity Service - Get All Content Sets Example</h2>
        <form>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="type">Entity Type:</label>
                    <select id="type">
                        <option value="default">Default</option>
                        <option value="print">Print</option>
                        <option value="email">Email</option>
                    </select>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

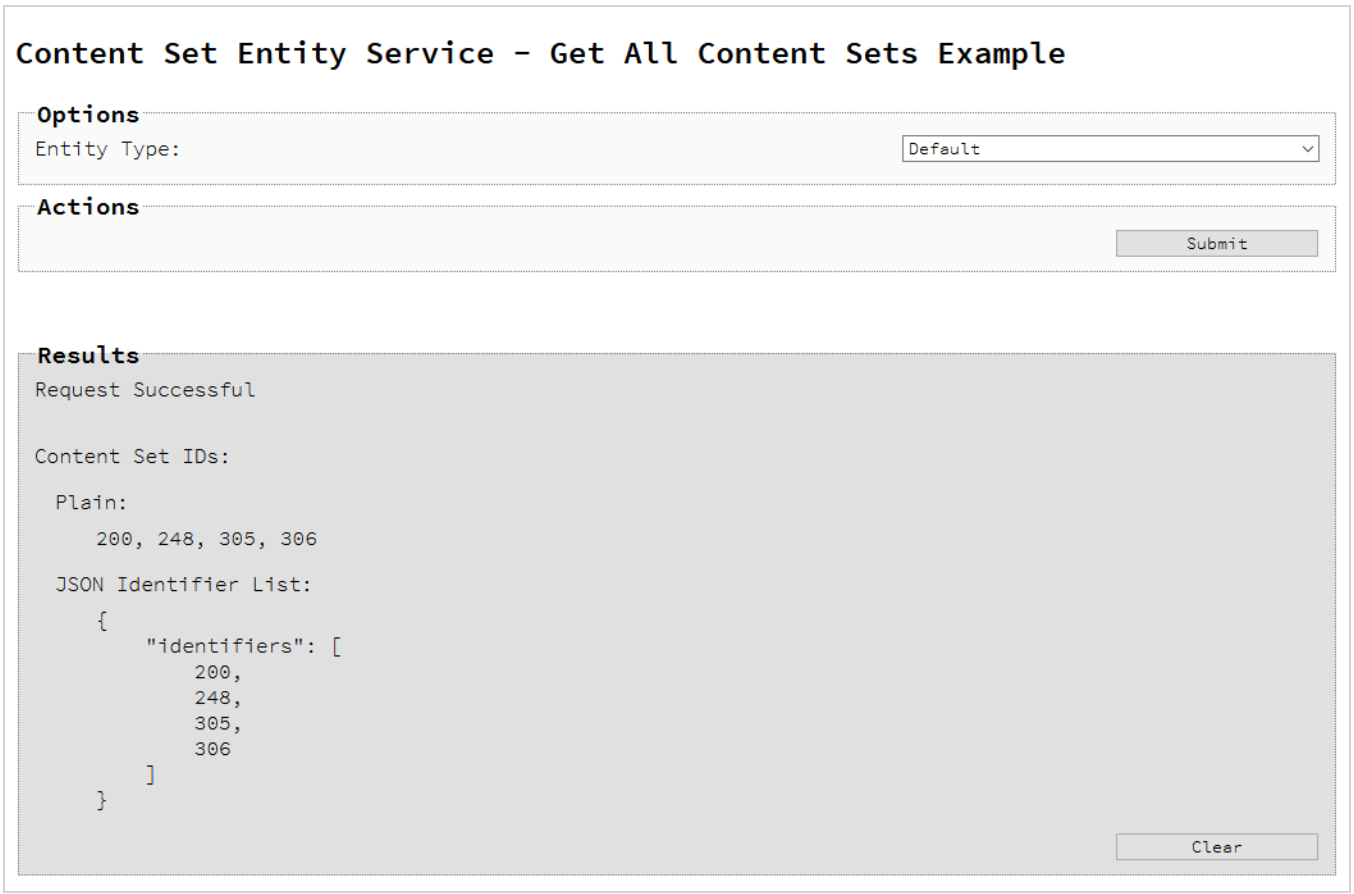
JavaScript/jQuery

cse-get-all-contentsets.js

```
/* Content Set Entity Service - Get All Content Sets Example */
(function ($, c) {
    "use strict";
    $(function () {
```

```
        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var type = $("#type").val();

            var settings = {
                type:   "GET",
                url:    "/rest/serverengine/entity/contentsets"
            };
            if (type !== "default") settings.data = { type: type };
            $.ajax(settings)
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayHeading("Content Set IDs");
                    c.displaySubResult("Plain", c.jsonIDListToPlain(response));
                    c.displaySubResult("JSON Identifier List", c.jsonPrettyPrint(response));
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply select the **Submit** button to request a list of the all the content sets currently contained within the server.

The resulting list will then be returned and displayed to the **Results** area in both Plain list and JSON Identifier List formats.

Further Reading

See the Content Set Entity Service page of the REST API Reference section for further detail.

Finding the Content Items in a Content Set

Problem

You want to obtain a list of all the previously created Content Items contained within a specific Content Set potentially for use in a Job Creation operation.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Set Entity REST service:

Get Content Items for Content Set	/rest/serverengine/entity/contentsets/{contentSetId}	*GET*

Example

HTML5

cse-get-contentitems.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get Content Items for Content Set Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cse-get-contentitems.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Set Entity Service - Get Content Items for Content Set Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="contentset">Content Set ID:</label>
                    <input id="contentset" type="text" placeholder="1234" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

cse-get-contentitems.js

```
/* Content Set Entity Service - Get Content Items for Content Set Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {
```

```
            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var contentSetId = $("#contentset").val();

            $.ajax({
                type:   "GET",
                url:    "/rest/serverengine/entity/contentsets/" + contentSetId
            })
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayHeading("Content Item IDs for Content Set '" + contentSetId + "'");
                    c.displaySubResult("Plain", c.jsonContentItemIDListToTable(response));
                    c.displaySubResult("JSON Content Item Identifier List", c.jsonPrettyPrint(response));
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Content Set Entity Service - Get Content Items for Content Set Example

Inputs

Content Set ID: 306

Actions

Submit

Results

Request Successful

Content Item IDs for Content Set '306':

Plain:

Content Item ID	Data Record ID
4001	4503
4002	4504

JSON Content Item Identifier List:


```
{
    "identifiers": [
        {
            "item": 4001,
            "record": 4503,
            "dataset": 158
        },
        {
            "item": 4002,
            "record": 4504,
            "dataset": 158
        }
    ]
}
```


Clear

Usage

To run the example simply enter the **Content Set ID** and select the **Submit** button to request a list of the all the content items contained within the specific content set in the server.

The resulting list will then be returned as a list of Content Item and Data Record ID pairs which will be displayed to the **Results** area in both Plain table and JSON Content Item Identifier List formats.

Further Reading

See the Content Set Entity Service page of the REST API Reference section for further detail.

Finding all the Job Sets in the Server

Problem

You want to obtain a list of all the previously created Job Sets contained in the PReS Connect Server potentially for use in a Output Creation operation.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Job Set Entity REST service:

Get All Job Sets	/rest/serverengine/entity/jobsets	*GET*

Example

HTML5

jse-get-all-jobsets.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get All Job Sets Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/jse-get-all-jobsets.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Job Set Entity Service - Get All Job Sets Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="submit">No Input Required</label>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

jse-get-all-jobsets.js

```
/* Job Set Entity Service - Get All Job Sets Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            $.ajax({
                type:   "GET",
```

```
                url:    "/rest/serverengine/entity/jobsets"
            })
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayHeading("Job Set IDs");
                    c.displaySubResult("Plain", c.jsonIDListToPlain(response));
                    c.displaySubResult("JSON Identifier List", c.jsonPrettyPrint(response));
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Usage

To run the example simply select the **Submit** button to request a list of the all the job sets currently contained within the server.

The resulting list will then be returned and displayed to the **Results** area in both Plain list and JSON Identifier List formats.

Further Reading

See the Job Set Entity Service page of the REST API Reference section for further detail.

Finding the Jobs in a Job Set

Problem

You want to obtain a list of all the previously created Jobs contained within a specific Job Set potentially for use in a Output Creation operation.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Job Set Entity REST service:

Get Jobs for Job Set	/rest/serverengine/entity/jobsets/{jobSetId}	*GET*

Example

HTML5

jse-get-jobs.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get Jobs for Job Set Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/jse-get-jobs.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Job Set Entity Service - Get Jobs for Job Set Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="jobset">Job Set ID:</label>
                    <input id="jobset" type="text" placeholder="1234" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

jse-get-jobs.js

```
/* Job Set Entity Service - Get Jobs for Job Set Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {
```

```
            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var jobSetId = $("#jobset").val();

            $.ajax({
                type:   "GET",
                url:    "/rest/serverengine/entity/jobsets/" + jobSetId
            })
                .done(function (response) {
                    c.displayStatus("Request Successful");
                    c.displayHeading("Job IDs for Job Set '" + jobSetId + "'");
                    c.displaySubResult("Plain", c.jsonIDListToPlain(response));
                    c.displaySubResult("JSON Identifier List", c.jsonPrettyPrint(response));
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Usage

To run the example simply enter the **Job Set ID** and select the **Submit** button to request a list of the all the jobs contained within the specific job set in the server.

The resulting list will then be returned and displayed to the **Results** area in both Plain list and JSON Identifier List formats.

Further Reading

See the Job Set Entity Service page of the REST API Reference section for further detail.

Working with the Workflow Services

This section consists of a number of pages covering various useful working examples:

1. Running a Data Mapping Operation
2. Running a Data Mapping Operation (Using JSON)
3. Running a Data Mapping Operation for PDF/VT File (to Data Set)
4. Running a Data Mapping Operation for PDF/VT File (to Content Set)
5. Running a Content Creation Operation for Print By Data Set
6. Running a Content Creation Operation for Print By Data Record (Using JSON)
7. "Running a Content Creation Operation for Print By Data (Using JSON)" on page 236
8. "Creating a Preview PDF for Print By Data" on page 242
9. "Creating a Preview PDF for Print By Data Record" on page 240
10. "Creating a Preview PDF for Print By Data (Using JSON)" on page 244
11. "Creating a Preview Image By Data Record (Using JSON)" on page 246
12. "Creating a Preview Image By Data (Using JSON)" on page 253
13. "Running a Content Creation Operation for Email By Data (Using JSON)" on page 267
14. Running a Content Creation Operation for Email By Data Record (Using JSON)
15. Creating Content for Web By Data Record
16. Creating Content for Web By Data Record (Using JSON)
17. Creating Content for Web By Data (Using JSON)
18. Running a Job Creation Operation By Content Set (Using JSON)
19. "Running a Job Creation Operation By Content Set with Runtime Parameters (Using JSON)" on page 293
20. Running an Output Creation Operation By Job Set
21. Running an Output Creation Operation By Job Set (Using JSON)
22. Running an Output Creation Operation By Job (Using JSON)
23. Running an All-In-One Operation (Using JSON)
24. "Running an All-In-One Operation with Adhoc Data" on page 324

See the Data Mapping Service, Content Creation Service, Content Creation (Email) Service, Content Creation (HTML) Service, Job Creation Service, Output Creation Service and All-In-One Service pages of the REST API Reference section for further detail.

Note: A complete listing including these examples can be found in the **index.html** file located at the root of the working example source code which contains links to all working examples.

Running a Data Mapping Operation

Problem

You want to run a data mapping operation to produce a Data Set using a data file and a data mapping configuration as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the data mapping operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Data Mapping REST service:

Process Data Mapping	/rest/serverengine/workflow/datamining/{configId}/{dataFileId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/datamining/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/datamining/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/datamining/cancel/{operationId}	*POST*

Example

HTML5

dm-process.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Data Mapping Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dm-process.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Data Mapping Service - Process Data Mapping Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">Data File ID/Name:</label>
                    <input id="datafile" type="text" placeholder="1234 or Filename" required>
                </div>
                <div>
                    <label for="datamapper">Data Mapping Configuration ID/Name:</label>
                    <input id="datamapper" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="validate">Validate Only:</label>
                    <input id="validate" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
```

```
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

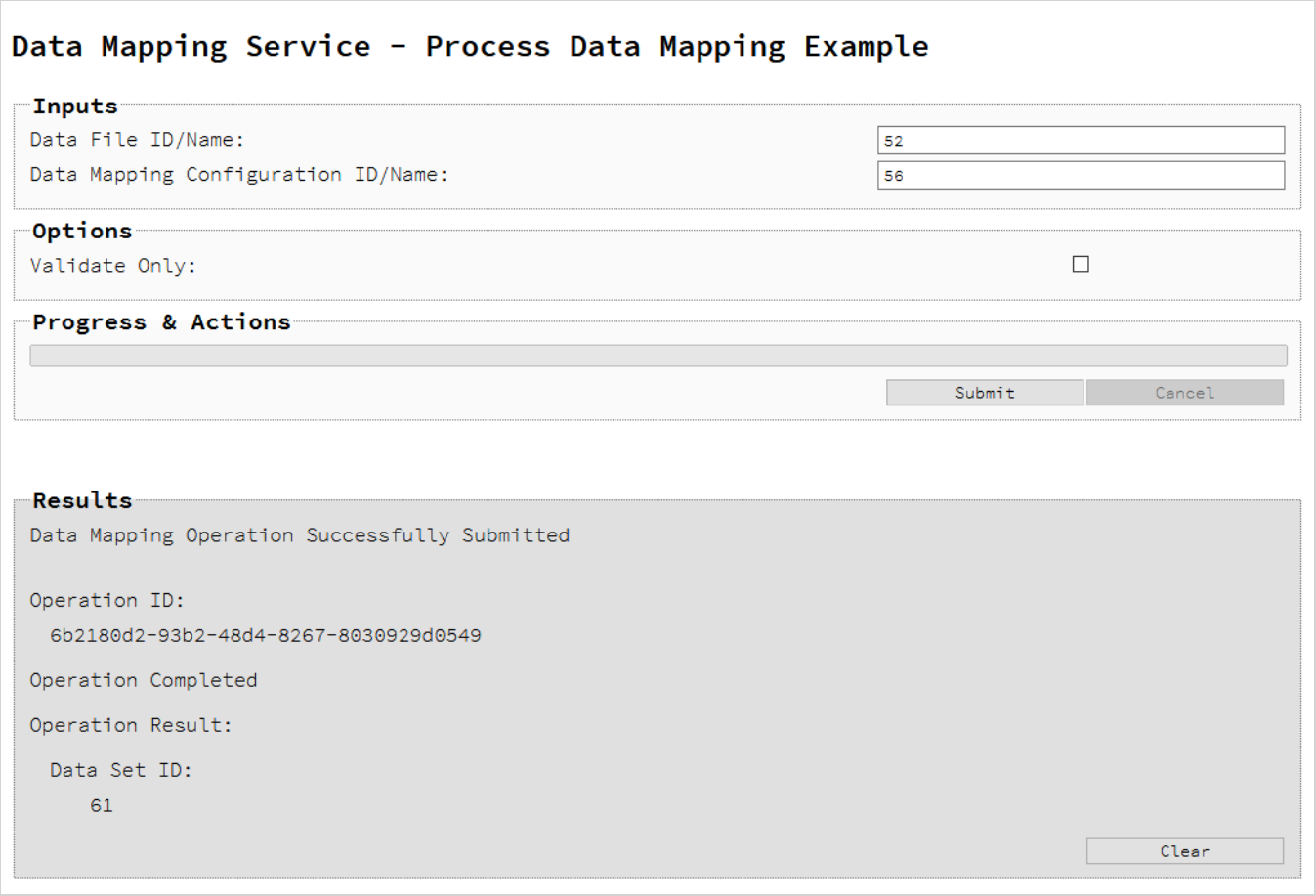
JavaScript/jQuery

dm-process.js

```
/* Data Mapping Service - Process Data Mapping Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var configId = $("#datamapper").val(),
                dataFileId = $("#datafile").val(),
                validate = $("#validate").prop("checked");

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        if (validate)
                            c.displaySubResult("JSON Data Mapping Validation Result",
                                c.jsonPrettyPrint(response));
                        else
                            c.displaySubResult("Data Set ID", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Data Mapping */
            $.ajax({
                type:   "POST",
                url:    "/rest/serverengine/workflow/datamining/" + configId + "/" + dataFileId +
```

```
                        "?validate=" + validate
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Data Mapping Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
                        if (operationId !== null) {

                            /* Get Progress of Operation */
                            $.ajax({
                                type:   "GET",
                                cache:  false,
                                url:    "/rest/serverengine/workflow/datamining/getProgress/" + operationId
                            })
                                .done(function (response, status, request) {

                                    if (response !== "done") {
                                        if (response !== progress) {
                                            progress = response;
                                            $progressBar.attr("value", progress);
                                        }
                                        setTimeout(getProgress, 1000);
                                    } else {
                                        $progressBar.attr("value", (progress = 100));
                                        c.displayInfo("Operation Completed");
                                        getFinalResult();
                                        operationId = null;
                                        setTimeout(function () {
                                            $progressBar.attr("value", 0);
                                            $submitButton.prop("disabled", false);
                                            $cancelButton.prop("disabled", true);
                                        }, 100);
                                    }
                                })
                                .fail(c.displayDefaultFailure);
                        }
                    };
                    getProgress();
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Managed File ID or Name** for your data file and your data mapping configuration (previously uploaded to the file store) into the appropriate text fields, and then check any options that you may require:

- **Validate Only** – Only validate the Data Mapping operation to check for mapping errors (no Data Set is created).

Lastly, select the **Submit** button to start the data mapping operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the data mapping operation has completed, the ID of the Data Set created will be returned and displayed to the **Results** area.

If the operation was to only validate the data mapping, then a JSON Data Mapping Validation Result will be returned and displayed instead.

Further Reading

See the Data Mapping Service page of the REST API Reference section for further detail.

Running a Data Mapping Operation (Using JSON)

Problem

You want to run a data mapping operation to produce a Data Set using a data file and a data mapping configuration as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the data mapping operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Data Mapping REST service:

Process Data Mapping (JSON)	/rest/serverengine/workflow/datamining/{configId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/datamining/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/datamining/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/datamining/cancel/{operationId}	*POST*

Example

HTML5

dm-process-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Data Mapping (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dm-process-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Data Mapping Service - Process Data Mapping (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">Data File ID/Name:</label>
                    <input id="datafile" type="text" placeholder="1234 or Filename" required>
                </div>
                <div>
                    <label for="datamapper">Data Mapping Configuration ID/Name:</label>
                    <input id="datamapper" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="validate">Validate Only:</label>
                    <input id="validate" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
```

```
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

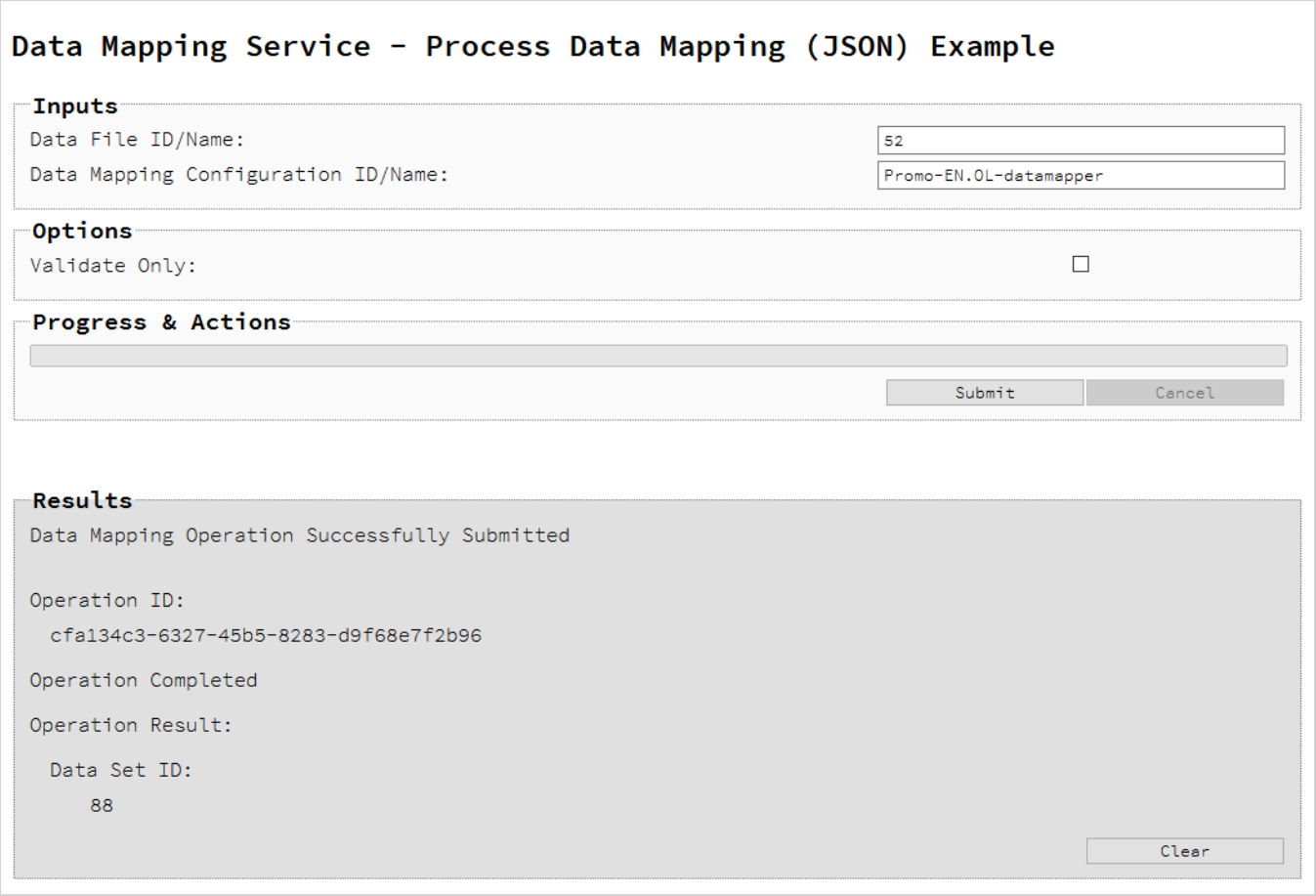
JavaScript/jQuery

dm-process-json.js

```
/* Data Mapping Service - Process Data Mapping (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var configId = $("#datamapper").val(),
                dataFileId = $("#datafile").val(),
                validate = $("#validate").prop("checked");

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        if (validate) {
                            c.displaySubResult("JSON Data Mapping Validation Result",
                                c.jsonPrettyPrint(response));
                        } else {
                            c.displaySubResult("Data Set ID", response);
                        }
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Data Mapping (JSON) */
            $.ajax({
                type:           "POST",
```

```
                url:            "/rest/serverengine/workflow/datamining/" + configId + "?validate=" + validate,
                data:           JSON.stringify(c.plainIDToJson(dataFileId)),
                contentType:    "application/json"
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Data Mapping Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
                        if (operationId !== null) {

                            /* Get Progress of Operation */
                            $.ajax({
                                type:   "GET",
                                cache:  false,
                                url:    "/rest/serverengine/workflow/datamining/getProgress/" + operationId
                            })
                                .done(function (response, status, request) {

                                    if (response !== "done") {
                                        if (response !== progress) {
                                            progress = response;
                                            $progressBar.attr("value", progress);
                                        }
                                        setTimeout(getProgress, 1000);
                                    } else {
                                        $progressBar.attr("value", (progress = 100));
                                        c.displayInfo("Operation Completed");
                                        getFinalResult();
                                        operationId = null;
                                        setTimeout(function () {
                                            $progressBar.attr("value", 0);
                                            $submitButton.prop("disabled", false);
                                            $cancelButton.prop("disabled", true);
                                        }, 100);
                                    }
                                })
                                .fail(c.displayDefaultFailure);
                        }
                    };
                    getProgress();
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Managed File ID or Name** for your data file and your data mapping configuration (previously uploaded to the file store) into the appropriate text fields, and then check any options that you may require:

- **Validate Only** – Only validate the Data Mapping operation to check for mapping errors (no Data Set is created).

Lastly, select the **Submit** button to start the data mapping operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the data mapping operation has completed, the ID of the Data Set created will be returned and displayed to the **Results** area.

If the operation was to only validate the data mapping, then a JSON Data Mapping Validation Result will be returned and displayed instead.

Further Reading

See the Data Mapping Service page of the REST API Reference section for further detail.

Running a Data Mapping Operation for PDF/VT File (to Data Set)

Problem

You want to run a data mapping operation to produce a Data Set using only a PDF/VT file as input.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the data mapping operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Data Mapping REST service:

Process Data Mapping (PDF/VT to Data Set)	/rest/serverengine/workflow/datamining/pdfvtds/{dataFileId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/datamining/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/datamining/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/datamining/cancel/{operationId}	*POST*

Example

HTML5

dm-process-pdfvt-dse.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Data Mapping (PDF/VT to Data Set) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dm-process-pdfvt-dse.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Data Mapping Service - Process Data Mapping (PDF/VT to Data Set) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">Data File ID/Name:</label>
                    <input id="datafile" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

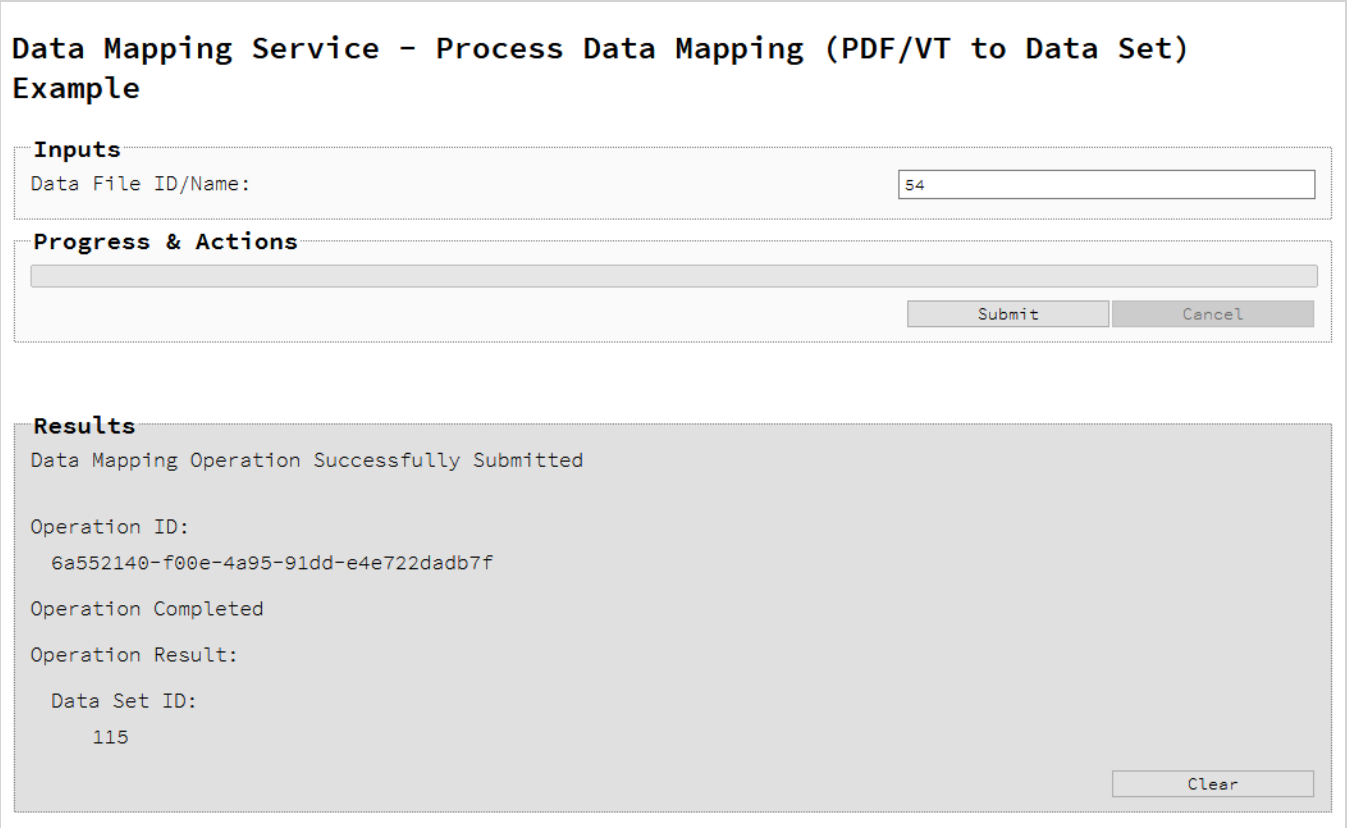
JavaScript/jQuery

dm-process-pdfvt-dse.js

```
/* Data Mapping Service - Process Data Mapping (PDF/VT to Data Set) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataFileId = $("#datafile").val();

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Data Set ID", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Data Mapping (PDF/VT to Data Set) */
            $.ajax({
                type:   "POST",
                url:    "/rest/serverengine/workflow/datamining/pdfvtds/" + dataFileId
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Data Mapping Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
                        if (operationId !== null) {

                            /* Get Progress of Operation */
                            $.ajax({
                                type:   "GET",
                                cache:  false,
                                url:    "/rest/serverengine/workflow/datamining/getProgress/" + operationId
                            })
                                .done(function (response, status, request) {

                                    if (response !== "done") {
```

```
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Managed File ID or Name** for your PDF/VT file (previously uploaded to the file store) into the appropriate text field, and then select the **Submit** button to start the

data mapping operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the data mapping operation has completed, the ID of the Data Set created will be returned and displayed to the **Results** area.

Further Reading

See the Data Mapping Service page of the REST API Reference section for further detail.

Running a Data Mapping Operation for PDF/VT File (to Content Set)

Problem

You want to run a data mapping operation to produce a Content Set using only a PDF/VT file as input.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the data mapping operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Data Mapping REST service:

Process Data Mapping (PDF/VT to Content Set)	/rest/serverengine/workflow/datamining/pdfvtcs/{dataFileId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/datamining/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/datamining/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/datamining/cancel/{operationId}	*POST*

Example

HTML5

dm-process-pdfvt-cse.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Data Mapping (PDF/VT to Content Set) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dm-process-pdfvt-cse.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Data Mapping Service - Process Data Mapping (PDF/VT to Content Set) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">Data File ID/Name:</label>
                    <input id="datafile" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

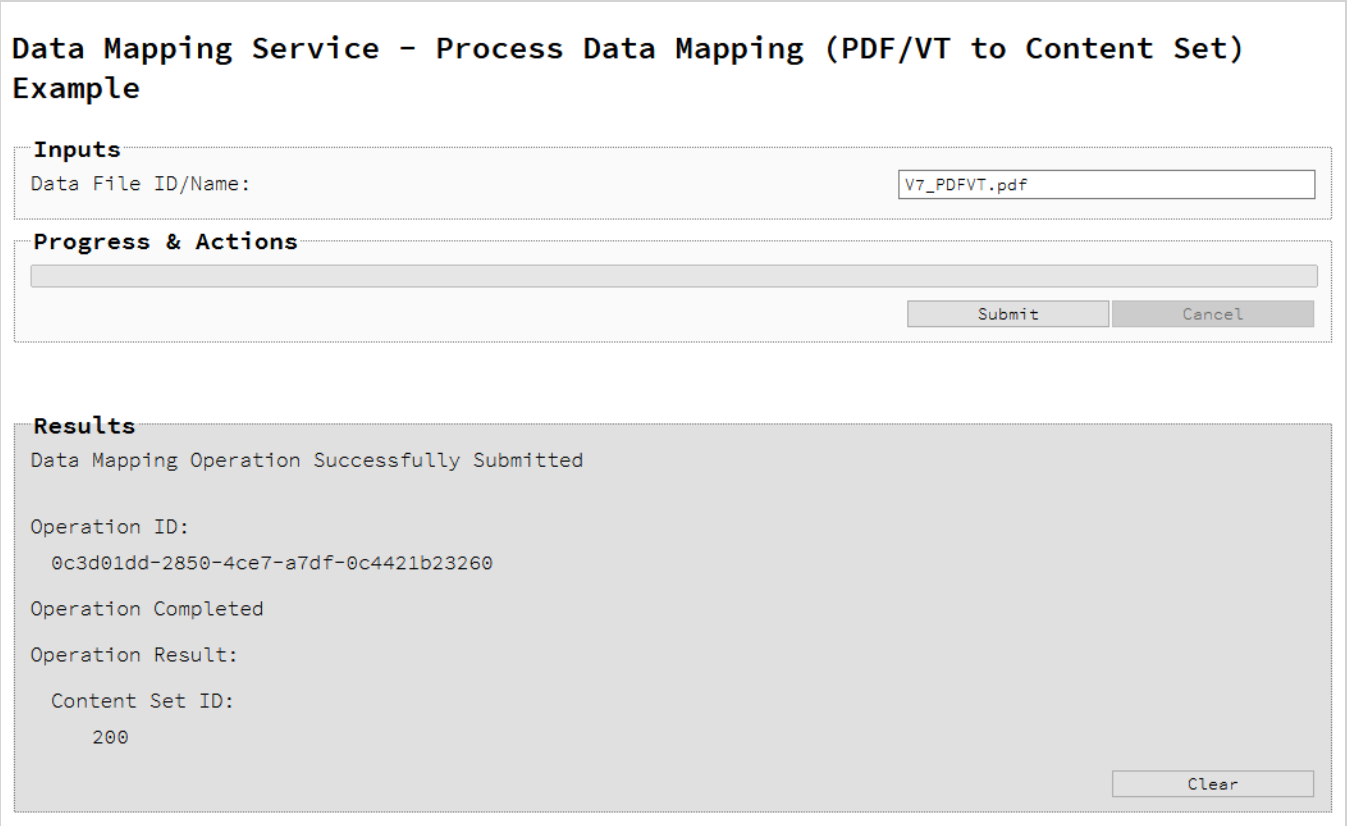
JavaScript/jQuery

dm-process-pdfvt-cse.js

```
/* Data Mapping Service - Process Data Mapping (PDF/VT to Content Set) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataFileId = $("#datafile").val();

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/datamining/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Content Set ID", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Data Mapping (PDF/VT to Content Set) */
            $.ajax({
                type:   "POST",
                url:    "/rest/serverengine/workflow/datamining/pdfvtcs/" + dataFileId
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Data Mapping Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
                        if (operationId !== null) {

                            /* Get Progress of Operation */
                            $.ajax({
                                type:   "GET",
                                cache:  false,
                                url:    "/rest/serverengine/workflow/datamining/getProgress/" + operationId
                            })
                                .done(function (response, status, request) {

                                    if (response !== "done") {
```

```
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Managed File ID or Name** for your PDF/VT file (previously uploaded to the file store) into the appropriate text field, and then select the **Submit** button to start the

data mapping operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the data mapping operation has completed, the ID of the Content Set created will be returned and displayed to the **Results** area.

Further Reading

See the Data Mapping Service page of the REST API Reference section for further detail.

Running a Content Creation Operation for Print By Data Set

Problem

You want to run a content creation operation to produce a Content Set using a template and an existing set of Data Records as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the content creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Content Creation REST service:

Process Content Creation	/rest/serverengine/workflow/contentcreation/{templateId}/{dataSetId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/contentcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/contentcreation/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/contentcreation/cancel/{operationId}	*POST*

Example

HTML5

cc-process-by-dse.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data Set) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-process-by-dse.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Process Content Creation (By Data Set) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="dataset">Data Set ID:</label>
                    <input id="dataset" type="text" placeholder="1234" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

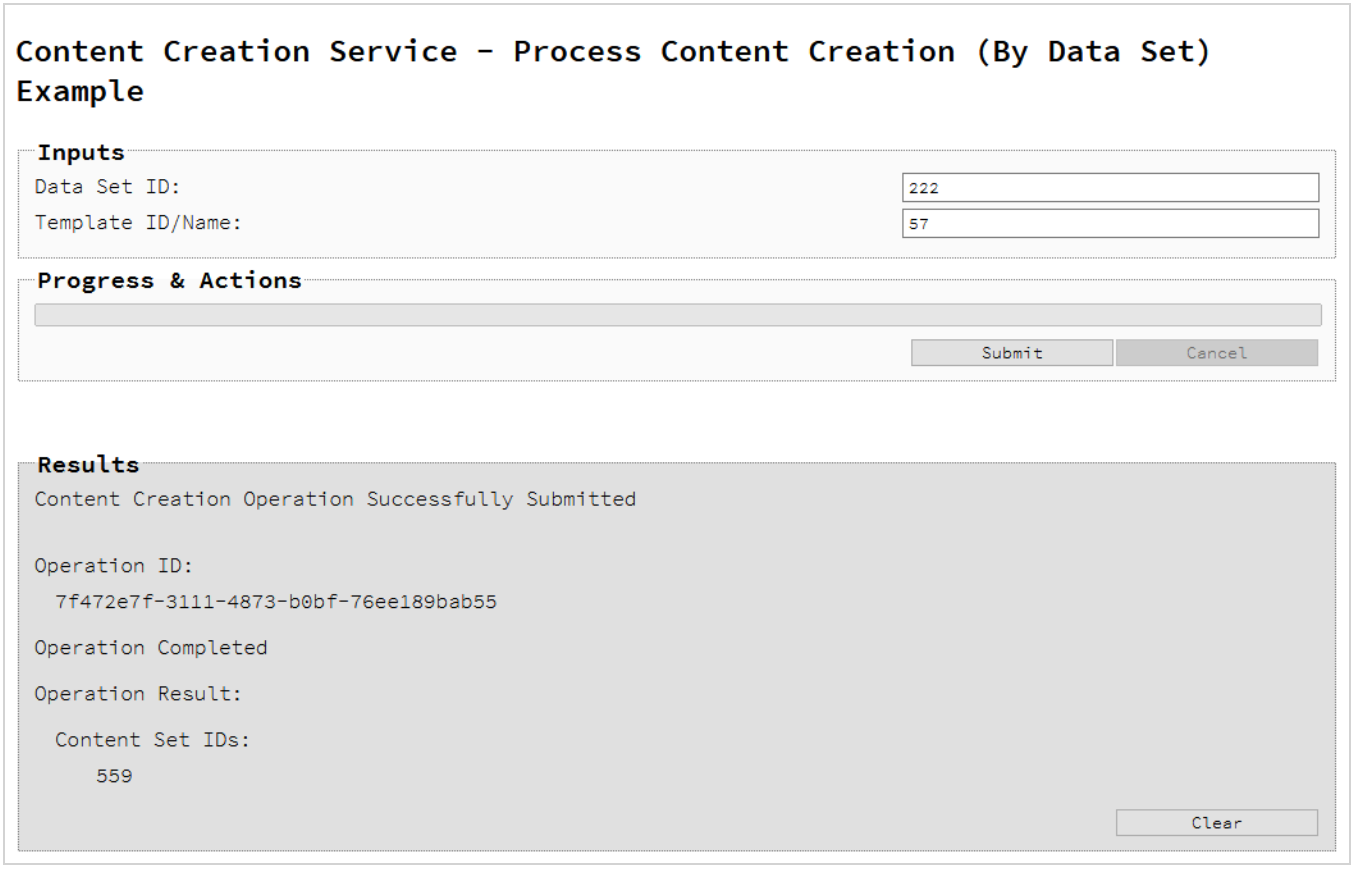
JavaScript/jQuery

cc-process-by-dse.js

```
/* Content Creation Service - Process Content Creation (By Data Set) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/contentcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataSetId = $("#dataset").val(),
                templateId = $("#template").val();

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/contentcreation/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Content Set IDs", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Content Creation (By Data Set) */
            $.ajax({
                type:   "POST",
                url:    "/rest/serverengine/workflow/contentcreation/" + templateId + "/" + dataSetId
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Content Creation Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
                        if (operationId !== null) {
```

```
                        /* Get Progress of Operation */
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/contentcreation/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Data Set ID** and the **Managed File ID or Name** of your template (previously uploaded to the file store) into the appropriate text fields, and then select the **Submit** button to start the content creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the content creation operation has completed, the IDs of the Content Sets created will be returned and displayed to the **Results** area.

Further Reading

See the Content Creation Service page of the REST API Reference section for further detail.

Running a Content Creation Operation for Print By Data Record (Using JSON)

Problem

You want to run a content creation operation to produce a Content Set using a template and an existing set of Data Records as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the content creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Content Creation REST service:

Process Content Creation (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/{templateId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/contentcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/contentcreation/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/contentcreation/cancel/{operationId}	*POST*

Example

HTML5

cc-process-by-dre-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data Record) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-process-by-dre-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Process Content Creation (By Data Record) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datarecords">Data Record ID(s):</label>
                    <input id="datarecords" type="text" placeholder="1234, 2345, 3456, ..." required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

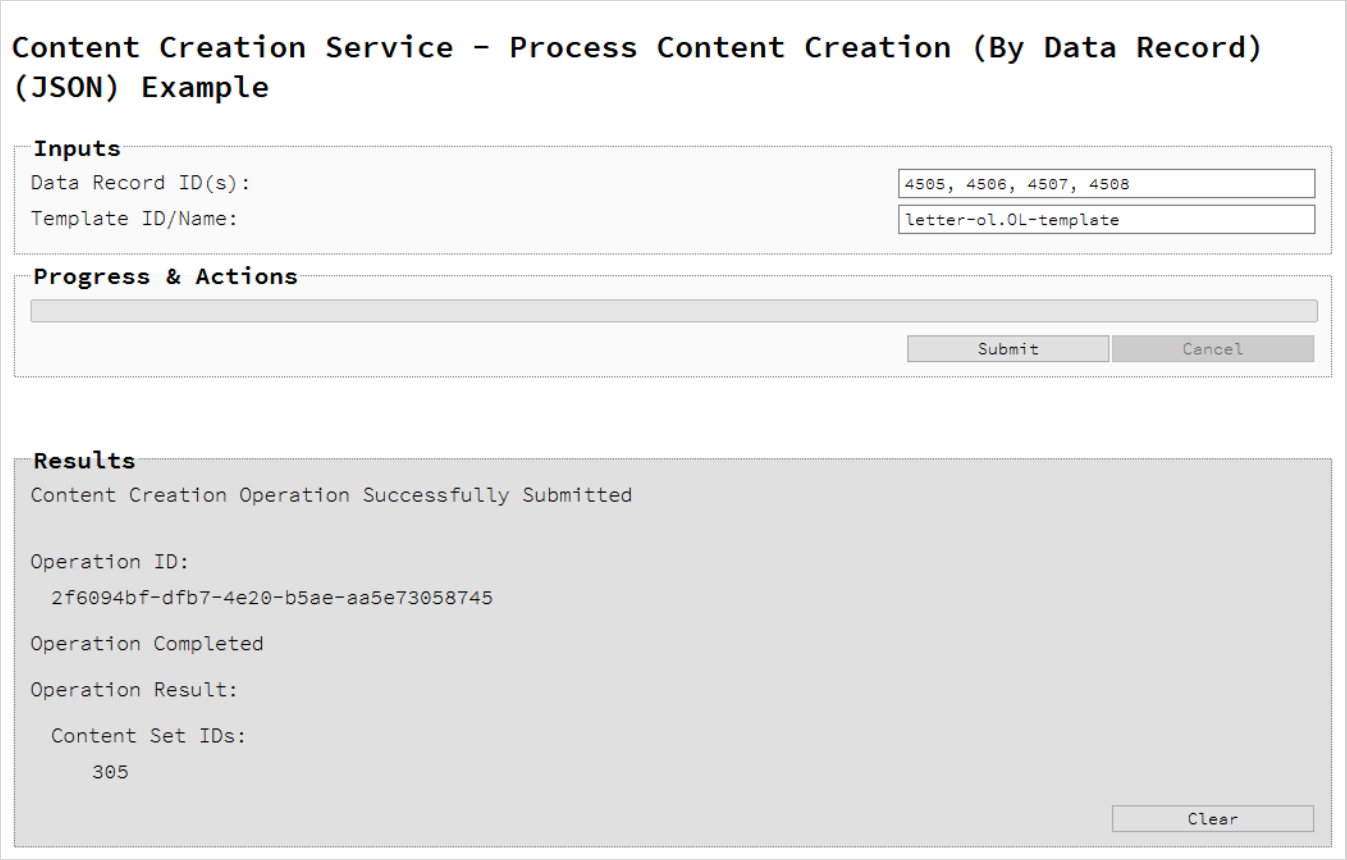
JavaScript/jQuery

cc-process-by-dre-json.js

```
/* Content Creation Service - Process Content Creation (By Data Record) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/contentcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataRecordIds = $("#datarecords").val(),
                templateId = $("#template").val();

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/contentcreation/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Content Set IDs", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Content Creation (By Data Record) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/" + templateId,
                data:           JSON.stringify(c.plainIDListToJson(dataRecordIds)),
                contentType:    "application/json"
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Content Creation Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
```

```
                    if (operationId !== null) {

                        /* Get Progress of Operation */
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/contentcreation/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter a comma delimited list of your **Data Record IDs** and the **Managed File ID or Name** of your template (previously uploaded to the File Store) into the appropriate text fields, and then select the **Submit** button to start the content creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the content creation operation has completed, the IDs of the Content Sets created will be returned and displayed to the **Results** area.

Further Reading

See the Content Creation Service page of the REST API Reference section for further detail.

Running a Content Creation Operation for Print By Data (Using JSON)

Problem

You want to run a content creation operation to produce a Content Set using a template and JSON data as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the content creation operation. There is also the option of canceling an operation during processing if required. These requests can be submitted via the Content Creation REST service:

Process Content Creation (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/{templateId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/contentcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/contentcreation/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/contentcreation/cancel/{operationId}	*POST*

Example

HTML5

cc-process-by-data-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-process-by-data-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Process Content Creation (By Data) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">JSON Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

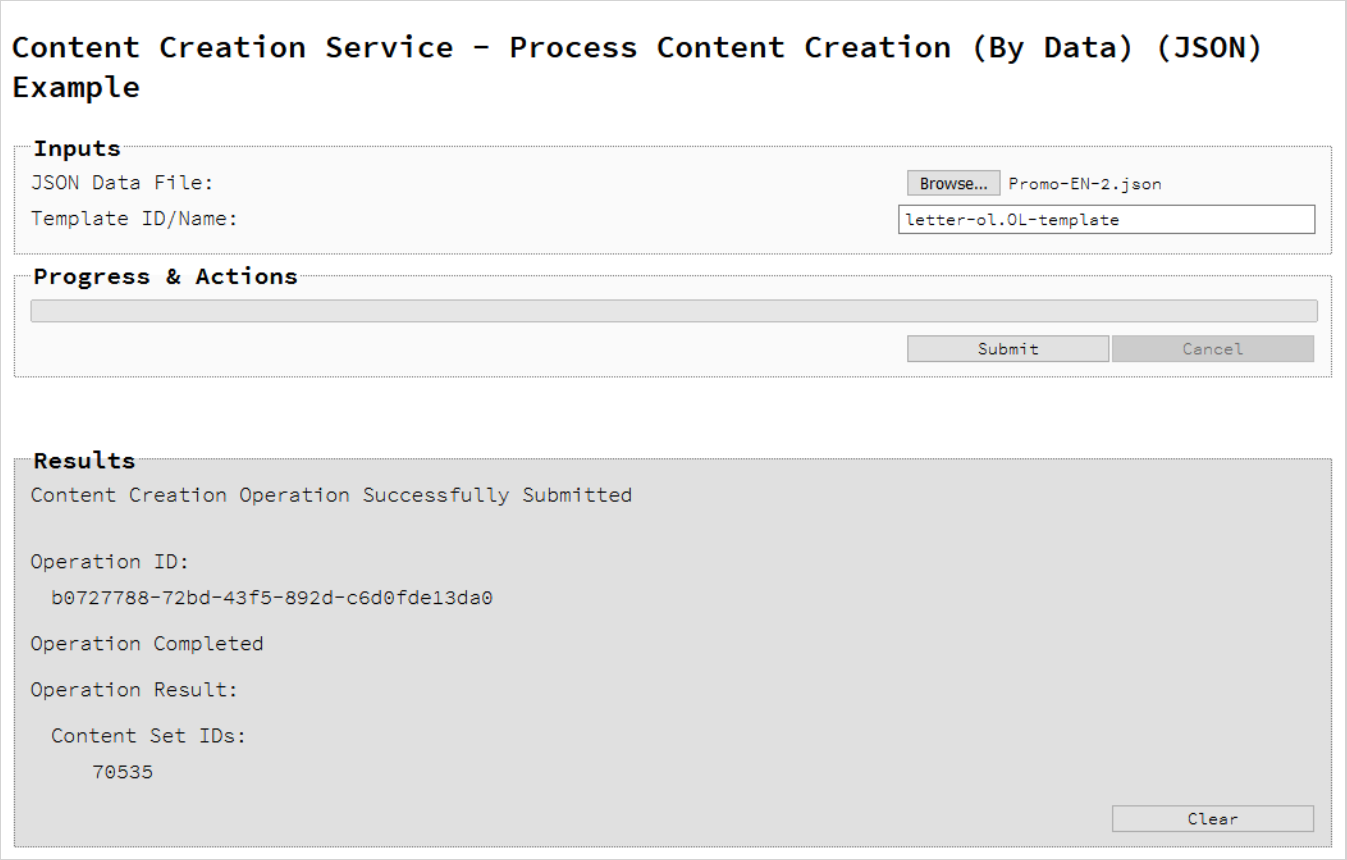
JavaScript/jQuery

cc-process-by-data-json.js

```
/* Content Creation Service - Process Content Creation (By Data) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null,
            jsonData = null;

        c.setupJsonDataFileInput($("#datafile"), function (data) { jsonData = data });

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/contentcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var templateId = $("#template").val();

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/contentcreation/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Content Set IDs", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Content Creation (By Data) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/" + templateId,
                data:           JSON.stringify({ data: jsonData }),
                contentType:    "application/json"
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Content Creation Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);
```

```
                var getProgress = function () {
                    if (operationId !== null) {

                        /* Get Progress of Operation */
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/contentcreation/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example you first need to use the Browse button to select an appropriate **JSON Data File** and then enter the Template **Managed File ID/Name** (previously uploaded to the File Store) into the appropriate text field as your inputs.

Select the **Submit** button to start the content creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the content creation operation has completed, the IDs of the Content Sets created will be returned and displayed to the **Results** area.

Further Reading

See the "Content Creation Service" on page 386 page of the REST API Reference section for further detail.

Creating a Preview PDF for Print By Data Record

Problem

You want to create a preview PDF using a template and an existing Data Record as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation REST service:

Create Preview PDF (By Data Record)	/rest/serverengine/workflow/contentcreation/pdfpreviewdirect	*GET*

Example

HTML5

cc-preview-pdf-by-dre.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Create Preview PDF (By Data Record) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-preview-pdf-by-dre.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Create Preview PDF (By Data Record) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datarecord">Data Record ID:</label>
                    <input id="datarecord" type="text" placeholder="1234" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

cc-preview-pdf-by-dre.js

```
/* Content Creation Service - Create Preview PDF (By Data Record) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {
```

```
            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var params = {
                dataRecordId: $("#datarecord").val(),
                templateId:   $("#template").val()
            };

            /* Create Preview PDF (By Data Record) */
            $.ajax({
                type:   "GET",
                url:    "/rest/serverengine/workflow/contentcreation/pdfpreviewdirect",
                data:   params
            })
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Usage

To run the example you need to select the specific **Data Record ID** and the Template **Managed File ID/Name** (previously uploaded to the File Store).

Once the inputs are entered, select the **Submit** button to create the preview PDF. When the response returns, a **Managed File ID** of the preview PDF (in the file store) will be displayed in the **Results** area.

Further Reading

See the "Content Creation Service" on page 386 page of the REST API Reference section for further detail.

Creating a Preview PDF for Print By Data

Problem

You want to create a preview PDF using a data file, data mapping configuration and a template as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation REST service:

Create Preview PDF (By Data)	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}/{dmConfigId}	*POST*

Example

HTML5

cc-preview-pdf-by-data.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Create Preview PDF (By Data) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-preview-pdf-by-data.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Create Preview PDF (By Data) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
                <div>
                    <label for="datamapper">Data Mapping Configuration ID/Name:</label>
                    <input id="datamapper" type="text" placeholder="1234 or Filename" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="persist">Persist Data Record:</label>
                    <input id="persist" type="checkbox" checked>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

cc-preview-pdf-by-data.js

```
/* Content Creation Service - Create Preview PDF (By Data) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var file        = $("#datafile")[0].files[0],
                dmConfigId  = $("#datamapper").val(),
                templateId  = $("#template").val(),
                persist     = $("#persist").prop("checked");

            /* Create Preview PDF (By Data) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/pdfpreview/"
                                    + templateId + "/" + dmConfigId + "?persist=" + persist,
                data:           file,
                processData:    false,
                contentType:    "application/octet-stream"
            })
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Usage

To run the example you first need to use the **Browse** button to select an appropriate **Data File** to use as an input using the selection dialog box.

Next, you need to enter the **Managed File ID or Name** of both your data mapping configuration and template (previously uploaded to the file store) into the appropriate text fields.

Lastly, you can specify the following option to use when creating the preview PDF:

- **Persist Data Records** – Create/persist data record entity in the server during the content creation process (intended for use with once-off jobs where the storage of the data record in the server is not required).

Once done, select the **Submit** button to create the preview PDF. When the response returns, a **Managed File ID** of the preview PDF (in the file store) will be displayed in the **Results** area.

Further Reading

See the "Content Creation Service" on page 386 page of the REST API Reference section for further detail.

Creating a Preview PDF for Print By Data (Using JSON)

Problem

You want to create a preview PDF using JSON data and a template as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation REST service:

Create Preview PDF (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}	*POST*

Example

HTML5

cc-preview-pdf-by-data-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Create Preview PDF (By Data) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-preview-pdf-by-data-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Create Preview PDF (By Data) (JSON) Example</h2>
        <form>
            <fieldset>
```

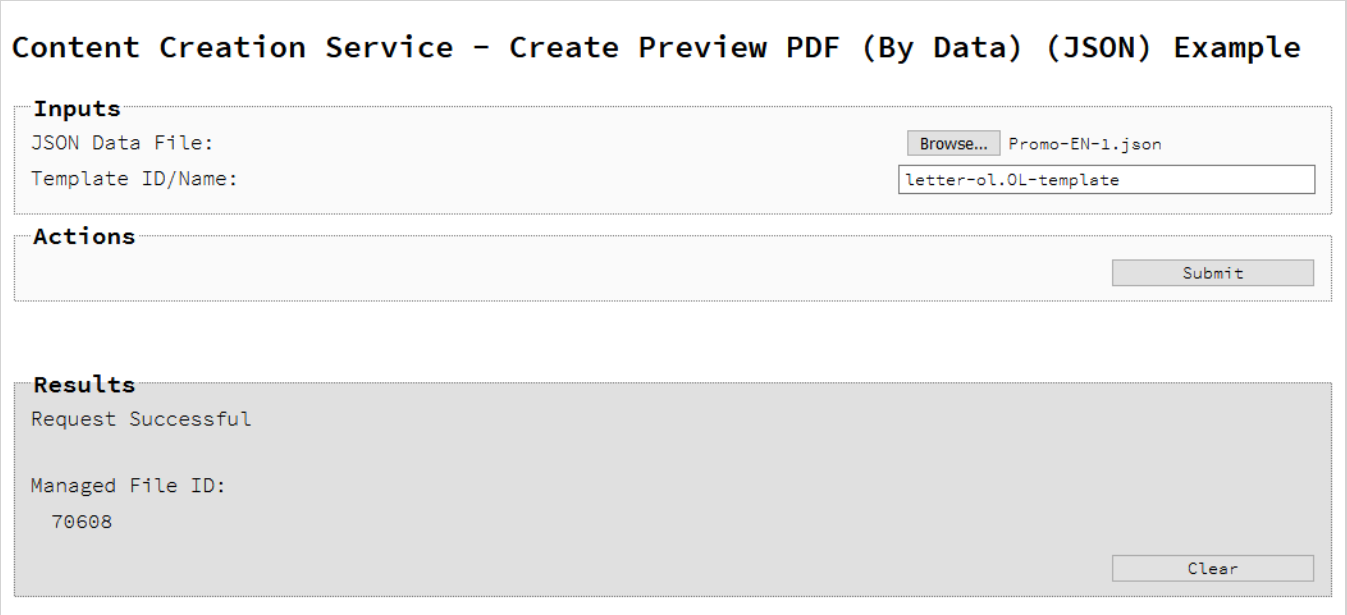
```
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">JSON Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

cc-preview-pdf-by-data-json.js

```
/* Content Creation Service - Create Preview PDF (By Data) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var jsonData = null;

        c.setupJsonDataFileInput($("#datafile"), function (data) { jsonData = data });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var templateId  = $("#template").val();

            /* Create Preview PDF (By Data) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/pdfpreview/" + templateId,
                data:           JSON.stringify({ data: jsonData }),
                contentType:    "application/json"
            })
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example you first need to use the **Browse** button to select an appropriate **JSON Data File** to use as an input using the selection dialog box.

Next, you need to enter the **Managed File ID or Name** of your template (previously uploaded to the file store) into the appropriate text field.

Once the inputs are selected/entered, select the **Submit** button to create the preview PDF. When the response returns, a **Managed File ID** of the preview PDF (in the file store) will be displayed in the **Results** area.

Further Reading

See the "Content Creation Service" on page 386 page of the REST API Reference section for further detail.

Creating a Preview Image By Data Record (Using JSON)

Problem

You want to create one or more preview images using a template and an existing Data Record as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation REST service:

Create Preview Image (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/imagepreview	*POST*

Example

HTML5

cc-preview-image-by-dre-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Create Preview Image (By Data Record) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-preview-image-by-dre-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Create Preview Image (By Data Record) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datarecord">Data Record ID:</label>
                    <input id="datarecord" type="text" placeholder="1234" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Image Parameters</legend>
                <div>
                    <label for="context">Context:</label>
                    <select id="context">
                        <option value="default">Default</option>
                        <option value="print">Print</option>
                        <option value="web">Web</option>
                        <option value="email">Email</option>
                    </select>
                </div>
                <div>
                    <label for="section">Section:</label>
                    <input id="section" type="text" placeholder="Section Name">
                </div>
                <div>
                    <label for="type">Type:</label>
                    <select id="type">
                        <option value="jpg">JPG</option>
                        <option value="jpeg">JPEG</option>
                        <option value="png">PNG</option>
                    </select>
                </div>
                <div>
                    <label for="quality">Quality:</label>
                    <input id="quality" type="number" min="0" max="100" value="100">
                </div>
                <div>
                    <label for="dpi">DPI:</label>
                    <input id="dpi" type="number" min="1" max="1200" value="96">
                </div>
                <div>
                    <label for="archive">Archive:</label>
                    <select id="archive">
                        <option value="default">Default</option>
                        <option value="true">True</option>
                        <option value="false">False</option>
                    </select>
                </div>
                <div>
                    <label for="bleed">Include Bleed:</label>
                    <input id="bleed" type="checkbox" value=false>
                </div>
                <div>
                    <label for="pages">Pages:</label>
```

```
                <input id="pages" type="text" placeholder="1, 2, 3-5, 6">
            </div>
            <div>
                <label for="viewPortWidth">Viewport Width:</label>
                <input id="viewPortWidth" type="number" min="0" value="1024">
            </div>
        </fieldset>
        <fieldset>
            <legend>Actions</legend>
            <div>
                <input id="submit" type="submit" value="Submit">
            </div>
        </fieldset>
    </form>
</body>
</html>
```

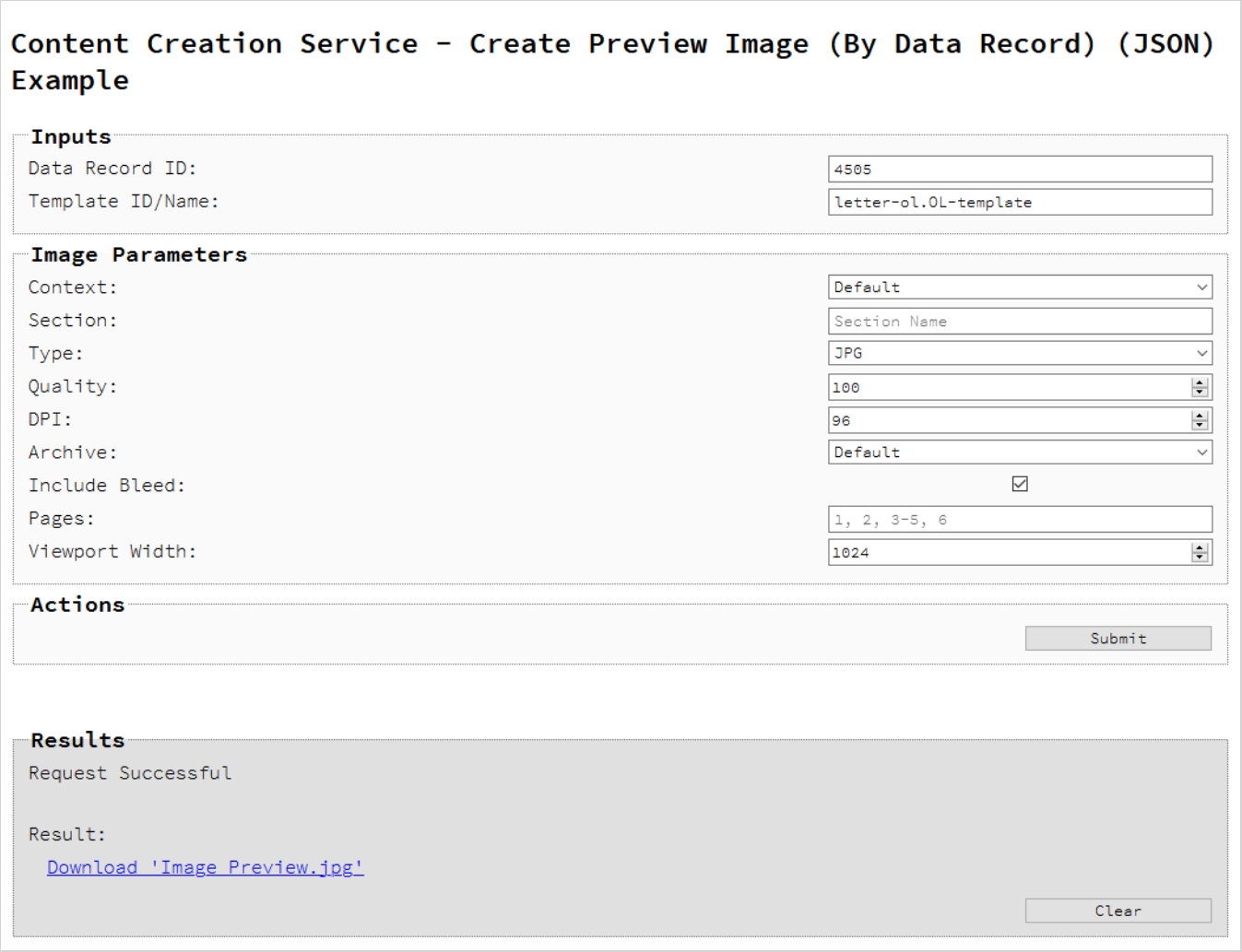
JavaScript/jQuery

cc-preview-image-by-dre-json.js

```
/* Content Creation Service - Create Preview Image (By Data Record) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $quality = $("#quality"),
            $archive = $("#archive"),
            $bleed = $("#bleed"),
            $pages = $("#pages"),
            $viewPortWidth = $("#viewPortWidth");

        $("#type")
            .on("change", function (event) {
                $quality.prop("disabled", ($(event.target).val() === "png"));
            })
            .trigger("change");

        $pages
            .on("change", function (event) {

                c.setCustomInputValidity(event.target,
                    function (value) {
                        return c.validateNumericRange(value, false, true);
                    },
                    "Invalid Pages value entered");

                if (event.target.validity.valid && $archive.val() === "false" &&
                    c.validateNumericRange(event.target.value, true, true)) {
                    $archive.val("true");
                }
            })
            .trigger("change");

        $("#context")
            .on("change", function (event) {

                var isDefault = ($(event.target).val() === "default"),
                    isPrint = ($(event.target).val() === "print");

                $pages.prop("disabled", (!isDefault && !isPrint));
                $bleed.prop("disabled", (!isDefault && !isPrint));
                $viewPortWidth.prop("disabled", (!isDefault && isPrint));
            })
            .trigger("change");

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataRecordId = $("#datarecord").val(),
                templateId = $("#template").val();

            /* Construct JSON Image Parameters */
```

```
            var config = {
                    "type":     $("#type").val(),
                    "dpi":      $("#dpi").val()
                },
                context = $("#context").val(),
                section = $("#section").val().trim(),
                archive = $archive.val();

            if (context !== "default") config.context = context;
            if (section.length) config.section = section;

            if (!$quality.prop("disabled")) config.quality = $quality.val();
            if (archive !== "default") config.archive = (archive === "true");
            if (!$bleed.prop("disabled")) config.bleed = $bleed.prop("checked");

            if (!$viewPortWidth.prop("disabled"))
                config.viewPortWidth = $viewPortWidth.val();

            if (!$pages.prop("disabled") && $pages.val().trim().length)
                config.pages = $pages.val();

            /* Create Preview Image (By Data Record) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/imagepreview/"
                                    + "?dataRecordId=" + dataRecordId + "&templateId=" + templateId,
                data:           JSON.stringify(config),
                contentType:    "application/json",
                dataType:       "file"
            })
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Result", c.dataToFileLink(response, "Image Preview"), false);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



OBJECTIF LUNE

www.objectiflune.com

2030 Pie-IX Blvd, Suite 500
Montreal (Quebec) Canada H1V 2C8

Jun 10, 2020

Mrs Lee Kang
Sailor
Westfield Industrial Co
42, Phillips Avenue
P.O. Box 13495
Westfield NB L3Z 9E2
Canada

Dear Mrs Kang,

Lorem ipsum dolor sit amet, consectetuer adipiscing elit. Nulla justo. Phasellus quis justo in est hendrerit blandit. Quisque ante lorem, sagittis sagittis, vestibulum vitae, nonummy eget, turpis. Vestibulum **PlanetPress Suite** eros urna, vehicula dapibus. Rutrum id consectetur vel.

Lorem ipsum dolor sit amet, **consectetur adipiscing elit**. Vivamus mollis erat ac orci aliquam sagittis. Maecenas elementum magna id leo vestibulum adipiscing. In facilisis fringilla magna, quis elementum odio sodales at. Lorum ipsum pellentesque accumsan tortor eu enim sagittis eleifend. Cras id ligula diam.

YOUR LICENSE: D4F70636B8

Quisque gravida, arcu vitae luctus feugiat, urna massa sollicitudin ligula, ac vehicula nisl urna et lorem. Sed rhoncus. Duis metus elit, iaculis et, tristique vitae, commodo vitae.

Sincerely,

Lorie Van Houtten

Lorie van Houtten
lvanhouten@ca.objectiflune.com
Sales Manager

Usage

To run the example you first select the appropriate **Data Record ID** and then add the Template **Managed File ID/Name** (previously uploaded to the File Store) into the appropriate text fields as your inputs.

Next, you can specify the following JSON Image Parameters to use when creating the preview image:

- **Context** – the context in the template to be used in the creation of the preview image (*Default* value is determined by the first context in the template).
- **Section** – the section within the context specified to be used in the creation of the preview image. If unspecified, then the default value is determined by the value of the **Context** image parameter specified.
  For the *Print* context this will be all enabled sections, for the *Email* and *Web* contexts this will be the default section.
- **Type** – the image type/format to be used in the creation of the preview image.
- **DPI** – the target resolution of the preview image in *dots per inch* (DPI).
- **Archive** – whether to return the resulting preview as a ZIP file/archive. The *Default value* is determined automatically by the number of image files in the preview output.

If the **Type** image parameter is set to a value of *PNG*, then the following image parameter can also be specified:

- **Quality** – the quality of the preview image (ranging in value from *0 - 100*).

If the **Context** image parameter is set to a value of *Print*, then the following image parameters can also be specified:

- **Include Bleed** – whether to include the bleed area in the preview image.
- **Pages** – the page range to be output in the preview image.
  If unspecified, then the default value is determined by the value of the **Archive** image parameter.
  If the **Archive** image parameter is set to a value of *False*, then the default value will be 1.
  If the **Archive** image parameter is set to a value of either *Default* or *True*, then the default value will be * (all pages).

Alternatively, if the **Context** image parameter is set to a value of *Email* or *Web*, then the following image parameter can also be specified:

- **Viewport Width** – the image width of the preview image in *pixels*.

Once the inputs and image parameters have been entered/specified, select the **Submit** button to create the preview image. When the response returns, a *download link* of the preview image (or images) will be displayed in the **Results** area.

Further Reading

See the "Content Creation Service" on page 386 page of the REST API Reference section for further detail.

Creating a Preview Image By Data (Using JSON)

Problem

You want to create one or more preview images using a template and JSON data as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation REST service:

Create Preview Image (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/imagepreview/{templateId}	*POST*

Example

HTML5

cc-preview-image-by-data-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Create Preview Image (By Data) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-preview-image-by-data-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Create Preview Image (By Data) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">JSON Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Image Parameters</legend>
                <div>
                    <label for="context">Context:</label>
                    <select id="context">
                        <option value="default">Default</option>
                        <option value="print">Print</option>
                        <option value="web">Web</option>
                        <option value="email">Email</option>
                    </select>
                </div>
                <div>
                    <label for="section">Section:</label>
                    <input id="section" type="text" placeholder="Section Name">
                </div>
                <div>
                    <label for="type">Type:</label>
                    <select id="type">
                        <option value="jpg">JPG</option>
                        <option value="jpeg">JPEG</option>
```

```
                    <option value="png">PNG</option>
                </select>
            </div>
            <div>
                <label for="quality">Quality:</label>
                <input id="quality" type="number" min="0" max="100" value="100">
            </div>
            <div>
                <label for="dpi">DPI:</label>
                <input id="dpi" type="number" min="1" max="1200" value="96">
            </div>
            <div>
                <label for="archive">Archive:</label>
                <select id="archive">
                    <option value="default">Default</option>
                    <option value="true">True</option>
                    <option value="false">False</option>
                </select>
            </div>
            <div>
                <label for="bleed">Include Bleed:</label>
                <input id="bleed" type="checkbox" value=false>
            </div>
            <div>
                <label for="pages">Pages:</label>
                <input id="pages" type="text" placeholder="1, 2, 3-5, 6">
            </div>
            <div>
                <label for="viewPortWidth">Viewport Width:</label>
                <input id="viewPortWidth" type="number" min="0" value="1024">
            </div>
        </fieldset>
        <fieldset>
            <legend>Actions</legend>
            <div>
                <input id="submit" type="submit" value="Submit">
            </div>
        </fieldset>
    </form>
</body>
</html>
```

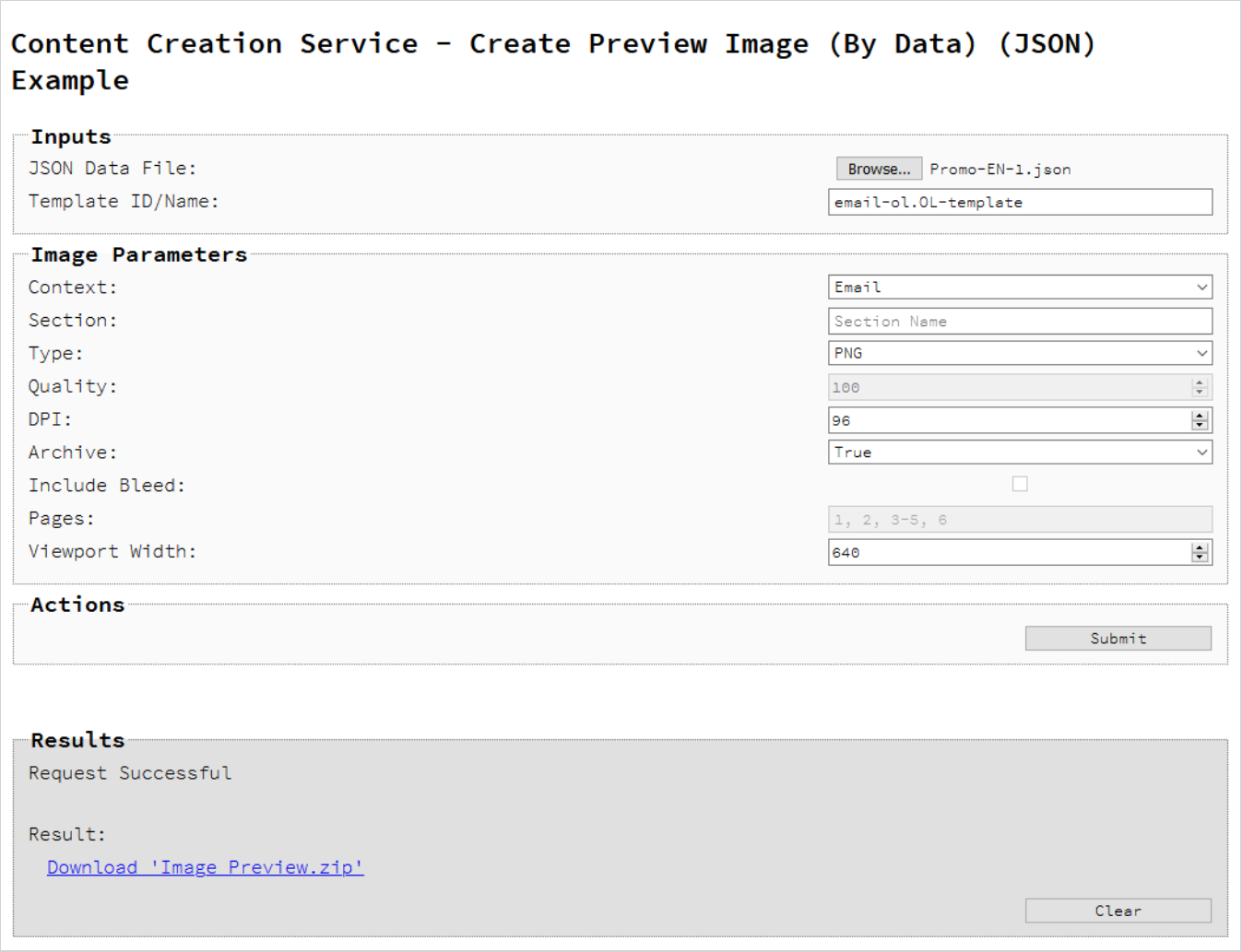
JavaScript/jQuery

cc-preview-image-by-data-json.js

```
/* Content Creation Service - Create Preview Image (By Data) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $quality = $("#quality"),
            $archive = $("#archive"),
            $bleed = $("#bleed"),
            $pages = $("#pages"),
            $viewPortWidth = $("#viewPortWidth"),
            jsonData = null;

        c.setupJsonDataFileInput($("#datafile"), function (data) { jsonData = data });

        $("#type")
            .on("change", function (event) {
                $quality.prop("disabled", ($(event.target).val() === "png"));
            })
            .trigger("change");

        $pages
            .on("change", function (event) {

                c.setCustomInputValidity(event.target,
                    function (value) {
                        return c.validateNumericRange(value, false, true);
                    },
                    "Invalid Pages value entered");
```

```
                if (event.target.validity.valid && $archive.val() === "false" &&
                    c.validateNumericRange(event.target.value, true, true)) {
                    $archive.val("true");
                }
            })
            .trigger("change");

        $("#context")
            .on("change", function (event) {

                var isDefault = ($(event.target).val() === "default"),
                    isPrint = ($(event.target).val() === "print");

                $pages.prop("disabled", (!isDefault && !isPrint));
                $bleed.prop("disabled", (!isDefault && !isPrint));
                $viewPortWidth.prop("disabled", (!isDefault && isPrint));
            })
            .trigger("change");

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var templateId = $("#template").val();

            /* Construct JSON Record Data List (with Image Parameters) */
            var config = {
                    "type":     $("#type").val(),
                    "dpi":      $("#dpi").val()
                },
                context = $("#context").val(),
                section = $("#section").val().trim(),
                archive = $archive.val();

            if (jsonData != null) config.data = jsonData;

            if (context !== "default") config.context = context;
            if (section.length) config.section = section;

            if (!$quality.prop("disabled")) config.quality = $quality.val();
            if (archive !== "default") config.archive = (archive === "true");
            if (!$bleed.prop("disabled")) config.bleed = $bleed.prop("checked");

            if (!$viewPortWidth.prop("disabled"))
                config.viewPortWidth = $viewPortWidth.val();

            if (!$pages.prop("disabled") && $pages.val().trim().length)
                config.pages = $pages.val();

            /* Create Preview Image (By Data) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/imagepreview/"
                                    + templateId,
                data:           JSON.stringify(config),
                contentType:    "application/json",
                dataType:       "file"
            })
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Result", c.dataToFileLink(response, "Image Preview"), false);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Lorem ipsum dolor sit amet, consectetur adipiscing elit.

OBJECTIF LUNE

Tired of bottlenecks?

Gain complete control!

Optimized production, distribution, and archival of transactional documents with automated workflows

PlanetPress Suite connects to any existing host server, ERP or application in order to generate reports and produce documents with total flexibility, with no complex programming or consultation fees.
With PlanetPress Suite you can:

- Document enhancement
- Automate distribution
- Mail preperation
- Automated archiving
- Document capture

Publish and output
Various output formats are also available making it possible to automatically publish documents to the web or to EDM systems while automating their distribution in the format preferred by recipients, electronically or not.

- Objectif Lune Support
- Contact US
- Software Download
- About PlanetPress Suite
- E-Learning center
- Partners Portal

Let's share ideas

Lorem ipsum dolor sit amet, Unsubscribe adipiscing elit.

Usage

To run the example you first need to use the **Browse** button to select an appropriate **JSON Data File** to use as an input using the selection dialog box.

Next, you need to enter the **Managed File ID or Name** of your template (previously uploaded to the file store) into the appropriate text field.

Lastly, you can specify the following JSON Image Parameters to use when creating the preview image

- **Context** – the context in the template to be used in the creation of the preview image (*Default* value is determined by the first context in the template).
- **Section** – the section within the context specified to be used in the creation of the preview image. If unspecified, then the default value is determined by the value of the **Context** image parameter specified.
  For the *Print* context this will be all enabled sections, for the *Email* and *Web* contexts this will be the default section.
- **Type** – the image type/format to be used in the creation of the preview image.
- **DPI** – the target resolution of the preview image in *dots per inch* (DPI)
- **Archive** – whether to return the resulting preview as a ZIP file/archive. The *Default value* is determined automatically by the number of image files in the preview output.

If the **Type** image parameter is set to a value of *PNG*, then the following image parameter can also be specified:

- **Quality** – the quality of the preview image (ranging in value from *0 - 100*).

If the **Context** image parameter is set to a value of *Print*, then the following image parameters can also be specified:

- **Include Bleed** – whether to include the bleed area in the preview image.
- **Pages** – the page range to be output in the preview image.
  If unspecified, then the default value is determined by the value of the **Archive** image parameter.
  If the **Archive** image parameter is set to a value of *False*, then the default value will be 1.
  If the **Archive** image parameter is set to a value of either *Default* or *True*, then the default value will be * (all pages).

Alternatively, if the **Context** image parameter is set to a value of *Email* or *Web*, then the following image parameter can also be specified:

- **Viewport Width** – the image width of the preview image in *pixels*.

Once the inputs and image parameters have been entered/specified, select the **Submit** button to create the preview image. When the response returns, a *download link* of the preview image (or images) will be displayed in the **Results** area.

Further Reading

See the "Content Creation Service" on page 386 page of the REST API Reference section for further detail.

Running a Content Creation Operation for Email By Data Record (Using JSON)

Problem

You want to run a content creation operation to create and potentially send email content using a template and an existing set of Data Records as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the content creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Content Creation (Email) REST service:

Process Content Creation (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/email/{templateId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/contentcreation/email/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/contentcreation/email/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/contentcreation/email/cancel/{operationId}	*POST*

Example

HTML5

cce-process-by-dre-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data Record) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cce-process-by-dre-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation (Email) Service - Process Content Creation (By Data Record) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datarecords">Data Record ID(s):</label>
                    <input id="datarecords" type="text" placeholder="1234, 2345, 3456, ..." required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Email Parameters</legend>
                <div>
                    <label for="section">Section:</label>
                    <input id="section" type="text" placeholder="Section Name">
                </div>
                <div>
                    <label for="sender">From:</label>
                    <input id="sender" type="text" placeholder="mailbox@domain.com" required>
                </div>
                <div>
                    <label for="sendername">From Name:</label>
```

```
                <input id="sendername" type="text" placeholder="From Name">
            </div>
            <div>
                <label for="host">Host:</label>
                <input id="host" type="text" placeholder="mail.domain.com:port">
            </div>
            <div>
                <label for="usesender">Use From as To Email Address:</label>
                <input id="usesender" type="checkbox" checked>
            </div>
            <div>
                <label for="attachpdf">Attach Print Context as PDF:</label>
                <select id="attachpdf">
                    <option value="default">Default</option>
                    <option value="true">True</option>
                    <option value="false">False</option>
                </select>
            </div>
            <div>
                <label for="attachweb">Attach Web Context as HTML:</label>
                <input id="attachweb" type="checkbox">
            </div>
            <div>
                <label for="eml">Add EML of Email:</label>
                <input id="eml" type="checkbox">
            </div>
        </fieldset>
        <fieldset>
            <legend>Email Security</legend>
            <div>
                <label for="useauth">Use Authentication:</label>
                <input id="useauth" type="checkbox" checked>
            </div>
            <div>
                <label for="starttls">Start TLS:</label>
                <input id="starttls" type="checkbox">
            </div>
            <div>
                <label for="username">Username:</label>
                <input id="username" type="text" placeholder="Username">
            </div>
            <div>
                <label for="password">Password:</label>
                <input id="password" type="password" placeholder="Password">
            </div>
        </fieldset>
        <fieldset>
            <legend>Progress &amp;amp; Actions</legend>
            <div>
                <progress value="0" max="100"></progress>
            </div>
            <div>
                <input id="cancel" type="button" value="Cancel" disabled>
                <input id="submit" type="submit" value="Submit">
            </div>
        </fieldset>
    </form>
</body>
</html>
```

JavaScript/jQuery

cce-process-by-dre-json.js

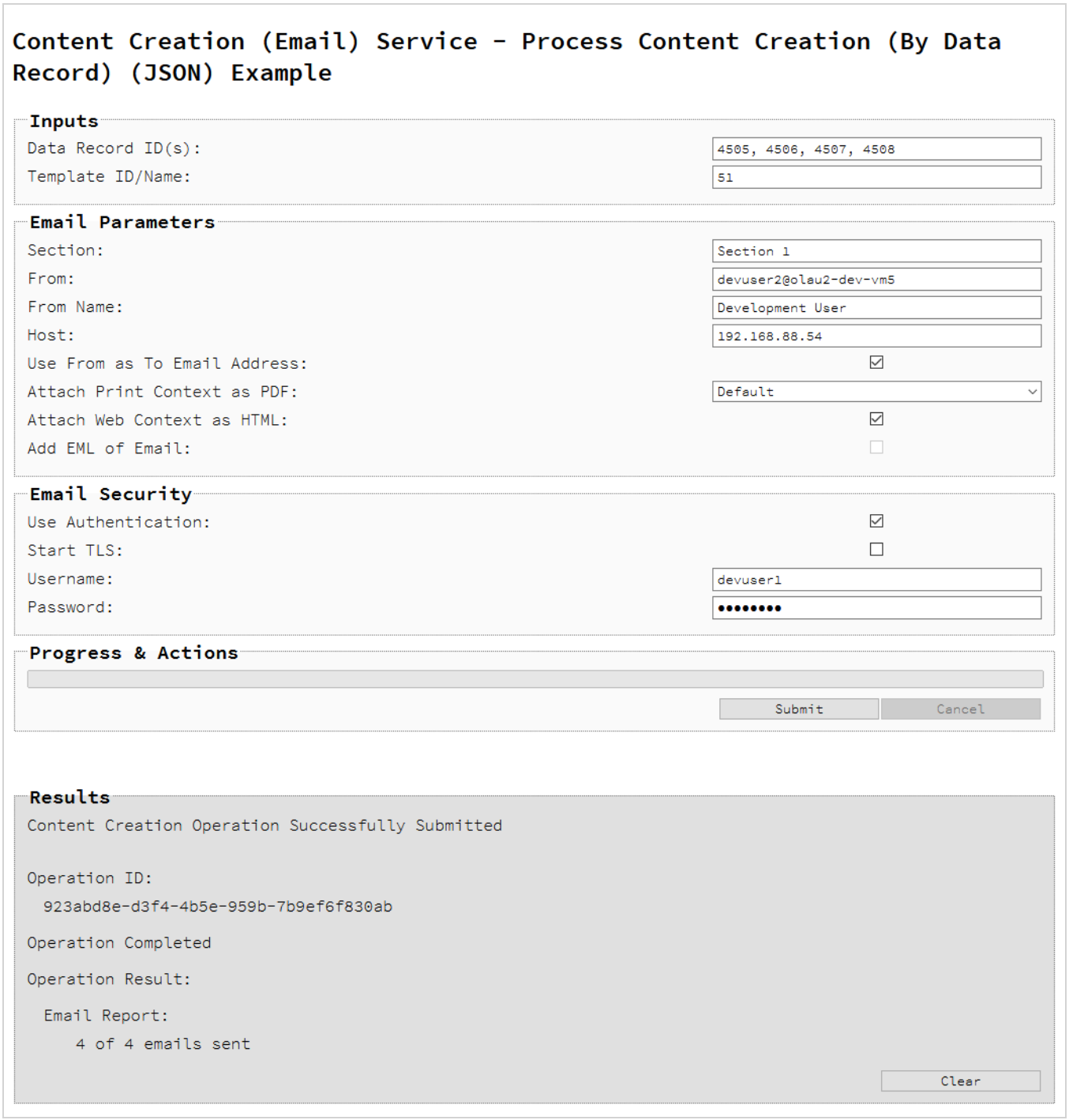
```
/* Content Creation (Email) Service - Process Content Creation (By Data Record) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $sender = $("#sender"),
            $senderName = $("#sendername"),
            $useSender = $("#usesender"),
            $host = $("#host"),
            $eml = $("#eml"),
            $useAuth = $("#useauth"),
```

```
    $startTLS = $("#starttls"),
    $username = $("#username"),
    $password = $("#password"),
    $submitButton = $("#submit"),
    $cancelButton = $("#cancel"),
    $progressBar = $("progress"),
    operationId = null;

$cancelButton.on("click", function () {
    if (operationId !== null) {

        /* Cancel an Operation */
        $.ajax({
            type:   "POST",
            url:    "/rest/serverengine/workflow/contentcreation/email/cancel/" + operationId
        })
            .done(function (response) {
                c.displayInfo("Operation Cancelled!");
                operationId = null;
                setTimeout(function () {
                    $progressBar.attr("value", 0);
                    $submitButton.prop("disabled", false);
                    $cancelButton.prop("disabled", true);
                }, 100);
            })
            .fail(c.displayDefaultFailure);
    }
});

$sender
    .on("change", function (event) {
        var sender = event.target.value.trim(),
            disabled = $(event.target).prop("disabled");
        $senderName.prop("disabled", (disabled || !sender.length));
        $useSender.prop("disabled", (disabled || !sender.length));
    })
    .trigger("change");

$host
    .on("change", function (event) {
        var host = event.target.value.trim();
        $useAuth
            .prop("disabled", !host.length)
            .trigger("change");
        $sender
            .prop("disabled", !host.length)
            .trigger("change");
        $eml
            .prop("disabled", host.length)
            .trigger("change");
    })
    .trigger("change");

$eml
    .on("change", function (event) {
        if (!$host.val().trim().length)
            $sender
                .prop("disabled", !$(event.target).prop("checked"))
                .trigger("change");
    })
    .trigger("change");

$useAuth
    .on("change", function (event) {
        var checked = ($(event.target).prop("checked")),
            disabled = ($(event.target).prop("disabled"));
        $.each([$startTLS, $username, $password], function (index, $element) {
            $element.prop("disabled", (disabled || !checked));
        });
    })
    .trigger("change");

$("form").on("submit", function (event) {

    event.preventDefault();
    if (!c.checkSessionValid()) return;

    var dataRecordIds = $("#datarecords").val(),
        templateId = $("#template").val(),
        section = $("#section").val().trim(),
        host = $host.val().trim();
```

```
var getFinalResult = function () {

    /* Get Result of Operation */
    $.ajax({
        type:   "POST",
        url:    "/rest/serverengine/workflow/contentcreation/email/getResult/" + operationId
    })
        .done(function (response, status, request) {
            c.displayHeading("Operation Result");
            if (host.length)
                c.displaySubResult("Email Report", response);
            else
                c.displaySubResult("JSON Email Output List",
                    c.jsonPrettyPrint(response));
        })
        .fail(c.displayDefaultFailure);
};

/* Construct JSON Identifier List (with Email Parameters) */
var config = {
        "identifiers": (c.plainIDListToJson(dataRecordIds)).identifiers
    },
    sender = $sender.val(),
    attachPdfPage = $("#attachpdf").val();

if (!$sender.prop("disabled") && sender.length) {
    config.sender = sender;
    var senderName = $senderName.val().trim();
    if (senderName.length) config.senderName = senderName;
    if ($useSender.prop("checked")) config.useSender = true;
}

if (host.length) {
    config.host = host;
    config.useAuth = $useAuth.prop("checked");
    if (config.useAuth) {
        if ($startTLS.prop("checked")) config.useStartTLS = true;
        config.user = $username.val();
        config.password = $password.val();
    }
} else {
    if ($eml.prop("checked")) config.eml = true;
}

if (attachPdfPage !== "default")
    config.attachPdfPage = (attachPdfPage === "true");
if ($("#attachweb").prop("checked"))
    config.attachWebPage = true;

/* Process Content Creation (By Data Record) (JSON) */
var settings = {
    type:           "POST",
    url:            "/rest/serverengine/workflow/contentcreation/email/" + templateId,
    data:           JSON.stringify(config),
    contentType:    "application/json"
};
if (section.length) settings.url += "?section=" + section;
$.ajax(settings)
    .done(function (response, status, request) {

        var progress = null;
        operationId = request.getResponseHeader("operationId");

        $submitButton.prop("disabled", true);
        $cancelButton.prop("disabled", false);

        c.displayStatus("Content Creation Operation Successfully Submitted");
        c.displayResult("Operation ID", operationId);

        var getProgress = function () {
            if (operationId !== null) {

                /* Get Progress of Operation */
                $.ajax({
                    type:   "GET",
                    cache:  false,
                    url:    "/rest/serverengine/workflow/contentcreation/email/getProgress/" + operationId
                })
                    .done(function (response, status, request) {

                        if (response !== "done") {
                            if (response !== progress) {
```

```
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example you first need to enter a comma delimited list of your **Data Record IDs** and the **Managed File ID or Name** of your template (previously uploaded to the file store) into the appropriate text fields as your inputs.

The example can then be configured (via the specification of email parameters) to run in one of two ways:

- Create email output to the Connect File Store.
- Create email output directly to a SMTP mail server.

By default, the example will create email output to the file store, but by specifying the **Host** email parameter the example can be run to create email output directly to a SMTP mail server.

Creating Email Output to the Connect File Store

When creating email output to the file store, the following email parameters can be specified for use with the content creation operation:

- **Add EML of Email** – Create an EML (E-Mail Message) file of the email for each record in the email output.

If the **Add EML of Email** field is checked, then the following email parameter must also be specified:

- **From** – the email address to be shown as the sender in the email output

Creating Email Output Directly to a SMTP Mail Server

When creating email output directly to a SMTP mail server, the following email parameters must be specified for use with the content creation operation:

- **Host** – the network address or name of your SMTP mail server through which the emails will be sent. If required, a server port value can also be specified.
- **From** – the email address to be shown as the sender in the email output.

Common to *both* forms of email output, the following email parameters can be specified for use with the content creation operation:

- **Section** – the section within the Email context of the template to use.
- **Attach Print Context as PDF** – if a Print context exists in the template, create its output as a PDF file and attach it to the email output (*Default* value depends on Template).
- **Attach Web Context as HTML** – if a Web context exists in the template, create output of its enabled sections (a single section by default) as HTML files and attach them to the email output.

Common to *both* forms of email output, if the **From** field is populated, then the following email parameters can also be specified:

- **From Name** – the name to be shown as the sender in the email output.
- **Use From as To Email Address** – use the sender address as the receiver address for all emails in the output.

Finally, if creating email output directly to a SMTP mail server, you need to specify how email security is to be used with the content creation operation:

- **Use Authentication** – if authentication is to be used with the mail server.
- **Start TLS** – if Transport Layer Security (TLS) is to be opportunistically used when sending emails.
- **Username** – the username to authenticate/login with.
- **Password** – the password to authenticate/login with.

Lastly, select the **Submit** button to start the content creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the content creation operation has completed, an operation result will be returned and displayed to the **Results** area.

If creating email output to the file store, a report of the emails successfully created will be returned in JSON Email Output List format. Alternatively, creating email output directly to a SMTP mail server will return a simple report of the emails successfully created and sent.

Further Reading

See the Content Creation (Email) Service page of the REST API Reference section for further detail.

Running a Content Creation Operation for Email By Data (Using JSON)

Problem

You want to run a content creation operation to create and potentially send email content using a template and JSON data as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the content creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Content Creation (Email) REST service:

Process Content Creation (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/email/{templateId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/contentcreation/email/getProgress/{operationId}	*GET*

Get Result of Operation	/rest/serverengine/workflow/contentcreation/email/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/contentcreation/email/cancel/{operationId}	*POST*

Example

HTML5

cce-process-by-data-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data Record) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cce-process-by-data-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation (Email) Service - Process Content Creation (By Data) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">JSON Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Email Parameters</legend>
                <div>
                    <label for="section">Section:</label>
                    <input id="section" type="text" placeholder="Section Name">
                </div>
                <div>
                    <label for="sender">From:</label>
                    <input id="sender" type="text" placeholder="mailbox@domain.com" required>
                </div>
                <div>
                    <label for="sendername">From Name:</label>
                    <input id="sendername" type="text" placeholder="From Name">
                </div>
                <div>
                    <label for="host">Host:</label>
                    <input id="host" type="text" placeholder="mail.domain.com:port">
                </div>
                <div>
                    <label for="usesender">Use From as To Email Address:</label>
                    <input id="usesender" type="checkbox" checked>
                </div>
                <div>
                    <label for="attachpdf">Attach Print Context as PDF:</label>
                    <select id="attachpdf">
                        <option value="default">Default</option>
                        <option value="true">True</option>
                        <option value="false">False</option>
                    </select>
                </div>
                <div>
                    <label for="attachweb">Attach Web Context as HTML:</label>
                    <input id="attachweb" type="checkbox">
                </div>
                <div>
                    <label for="eml">Add EML of Email:</label>
                    <input id="eml" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Email Security</legend>
                <div>
```

```
                    <label for="useauth">Use Authentication:</label>
                    <input id="useauth" type="checkbox" checked>
                </div>
                <div>
                    <label for="starttls">Start TLS:</label>
                    <input id="starttls" type="checkbox">
                </div>
                <div>
                    <label for="username">Username:</label>
                    <input id="username" type="text" placeholder="Username">
                </div>
                <div>
                    <label for="password">Password:</label>
                    <input id="password" type="password" placeholder="Password">
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

cce-process-by-data-json.js

```
/* Content Creation (Email) Service - Process Content Creation (By Data) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $sender = $("#sender"),
            $senderName = $("#sendername"),
            $useSender = $("#usesender"),
            $host = $("#host"),
            $eml = $("#eml"),
            $useAuth = $("#useauth"),
            $startTLS = $("#starttls"),
            $username = $("#username"),
            $password = $("#password"),
            $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null,
            jsonData = null;

        c.setupJsonDataFileInput($("#datafile"), function (data) { jsonData = data });

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/contentcreation/email/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
```

```
});

$sender
    .on("change", function (event) {
        var sender = event.target.value.trim(),
            disabled = $(event.target).prop("disabled");
        $senderName.prop("disabled", (disabled || !sender.length));
        $useSender.prop("disabled", (disabled || !sender.length));
    })
    .trigger("change");

$host
    .on("change", function (event) {
        var host = event.target.value.trim();
        $useAuth
            .prop("disabled", !host.length)
            .trigger("change");
        $sender
            .prop("disabled", !host.length)
            .trigger("change");
        $eml
            .prop("disabled", host.length)
            .trigger("change");
    })
    .trigger("change");

$eml
    .on("change", function (event) {
        if (!$host.val().trim().length)
            $sender
                .prop("disabled", !$(event.target).prop("checked"))
                .trigger("change");
    })
    .trigger("change");

$useAuth
    .on("change", function (event) {
        var checked = ($(event.target).prop("checked")),
            disabled = ($(event.target).prop("disabled"));
        $.each([$startTLS, $username, $password], function (index, $element) {
            $element.prop("disabled", (disabled || !checked));
        });
    })
    .trigger("change");

$("form").on("submit", function (event) {

    event.preventDefault();
    if (!c.checkSessionValid()) return;

    var templateId = $("#template").val(),
        section = $("#section").val().trim(),
        host = $host.val().trim();

    var getFinalResult = function () {

        /* Get Result of Operation */
        $.ajax({
            type:   "POST",
            url:    "/rest/serverengine/workflow/contentcreation/email/getResult/" + operationId
        })
            .done(function (response, status, request) {
                c.displayHeading("Operation Result");
                if (host.length)
                    c.displaySubResult("Email Report", response);
                else
                    c.displaySubResult("JSON Email Output List",
                        c.jsonPrettyPrint(response));
            })
            .fail(c.displayDefaultFailure);
    };

    /* Construct JSON Record Data List (with Email Parameters) */
    var config = {
            "data": jsonData
        },
        sender = $sender.val(),
        attachPdfPage = $("#attachpdf").val();

    if (!$sender.prop("disabled") && sender.length) {
        config.sender = sender;
        var senderName = $senderName.val().trim();
```

```
            if (senderName.length) config.senderName = senderName;
            if ($useSender.prop("checked")) config.useSender = true;
        }

        if (host.length) {
            config.host = host;
            config.useAuth = $useAuth.prop("checked");
            if (config.useAuth) {
                if ($startTLS.prop("checked")) config.useStartTLS = true;
                config.user = $username.val();
                config.password = $password.val();
            }
        } else {
            if ($eml.prop("checked")) config.eml = true;
        }

        if (attachPdfPage !== "default")
            config.attachPdfPage = (attachPdfPage === "true");
        if ($("#attachweb").prop("checked"))
            config.attachWebPage = true;

        /* Process Content Creation (By Data) (JSON) */
        var settings = {
            type:           "POST",
            url:            "/rest/serverengine/workflow/contentcreation/email/" + templateId,
            data:           JSON.stringify(config),
            contentType:    "application/json"
        };
        if (section.length) settings.url += "?section=" + section;
        $.ajax(settings)
            .done(function (response, status, request) {

                var progress = null;
                operationId = request.getResponseHeader("operationId");

                $submitButton.prop("disabled", true);
                $cancelButton.prop("disabled", false);

                c.displayStatus("Content Creation Operation Successfully Submitted");
                c.displayResult("Operation ID", operationId);

                var getProgress = function () {
                    if (operationId !== null) {

                        /* Get Progress of Operation */
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/contentcreation/email/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output

Content Creation (Email) Service - Process Content Creation (By Data) (JSON) Example

Inputs

JSON Data File: Browse... Promo-EN-1.json

Template ID/Name: email-ol.OL-template

Email Parameters

Section: Section 2

From: devuser3@olau2-dev-vm5

From Name: Development User

Host: mail.domain.com:port

Use From as To Email Address: ☐

Attach Print Context as PDF: False

Attach Web Context as HTML: ☐

Add EML of Email: ☑

Email Security

Use Authentication: ☑

Start TLS: ☐

Username: Username

Password: Password

Progress & Actions

Submit Cancel

Results

Content Creation Operation Successfully Submitted

Operation ID:

bc9c833c-fed5-4496-ac68-b3a63699e926

Operation Completed

Operation Result:

JSON Email Output List:


```
{
    "messages": [
        {
            "attachments": [
                {
                    "name": "att820ca11a-8e09-4779-94a1-352cefff6789.png",
                    "disposition": "inline"
                },
                {
                    "name": "att01c61954-9cec-4e0d-8e86-35b840282836.png",
                    "disposition": "inline"
                },
                {
                    "name": "att81bff3ac-8f13-481e-95a9-5e16b8b02111.jpg",
                    "disposition": "inline"
                },
                {
                    "name": "att9733fd71-4f19-4657-9b20-8f952f9538fa.png",
                    "disposition": "inline"
                },
                {
                    "name": "attfd43c109-c653-4f9c-ade7-a38e1014aa5b.png",
                    "disposition": "inline"
                },
```

Usage

To run the example you first need to use the **Browse** button to select an appropriate **JSON Data File** and then enter the **Managed File ID\Name** of your template (previously uploaded to the file store) into the appropriate text fields as your inputs.

The example can then be configured (via the specification of email parameters) to run in one of two ways:

- Create email output to the Connect File Store.
- Create email output directly to a SMTP mail server.

By default, the example will create email output to the file store, but by specifying the **Host** email parameter the example can be run to create email output directly to a SMTP mail server.

Creating Email Output to the Connect File Store

When creating email output to the file store, the following email parameters can be specified for use with the content creation operation:

- **Add EML of Email** – Create an EML (E-Mail Message) file of the email for each record in the email output.

If the **Add EML of Email** field is checked, then the following email parameter must also be specified:

- **From** – the email address to be shown as the sender in the email output.

Creating Email Output Directly to a SMTP Mail Server

When creating email output directly to a SMTP mail server, the following email parameters must be specified for use with the content creation operation:

- **Host** – the network address or name of your SMTP mail server through which the emails will be sent. If required, a server port value can also be specified.
- **From** – the email address to be shown as the sender in the email output.

Common to *both* forms of email output, the following email parameters can be specified for use with the content creation operation:

- **Section** – the section within the Email context of the template to use.
- **Attach Print Context as PDF** – if a Print context exists in the template, create its output as a PDF file and attach it to the email output (*Default* value depends on Template).
- **Attach Web Context as HTML** – if a Web context exists in the template, create output of its enabled sections (a single section by default) as HTML files and attach them to the email output.

Common to *both* forms of email output, if the **From** field is populated, then the following email parameters can also be specified:

- **From Name** – the name to be shown as the sender in the email output.
- **Use From as To Email Address** – use the sender address as the receiver address for all emails in the output.

Finally, if creating email output directly to a SMTP mail server, you need to specify how email security is to be used with the content creation operation:

- **Use Authentication** – if authentication is to be used with the mail server.
- **Start TLS** – if Transport Layer Security (TLS) is to be opportunistically used when sending emails.
- **Username** – the username to authenticate/login with.
- **Password** – the password to authenticate/login with.

Lastly, select the **Submit** button to start the content creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the content creation operation has completed, an operation result will be returned and displayed to the **Results** area.

If creating email output to the file store, a report of the emails successfully created will be returned in JSON Email Output List format. Alternatively, creating email output directly to a SMTP mail server will return a simple report of the emails successfully created and sent.

Further Reading

See the Content Creation (Email) Service page of the REST API Reference section for further detail.

Creating Content for Web By Data Record

Problem

You want to create and retrieve web content using a template and an existing Data Record as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation (HTML) REST service:

Process Content Creation (By Data Record)	/rest/serverengine/workflow/contentcreation/html/{templateId}/{dataRecordId: [0-9]+}	*GET*

Example

HTML5

cch-process-by-dre.html

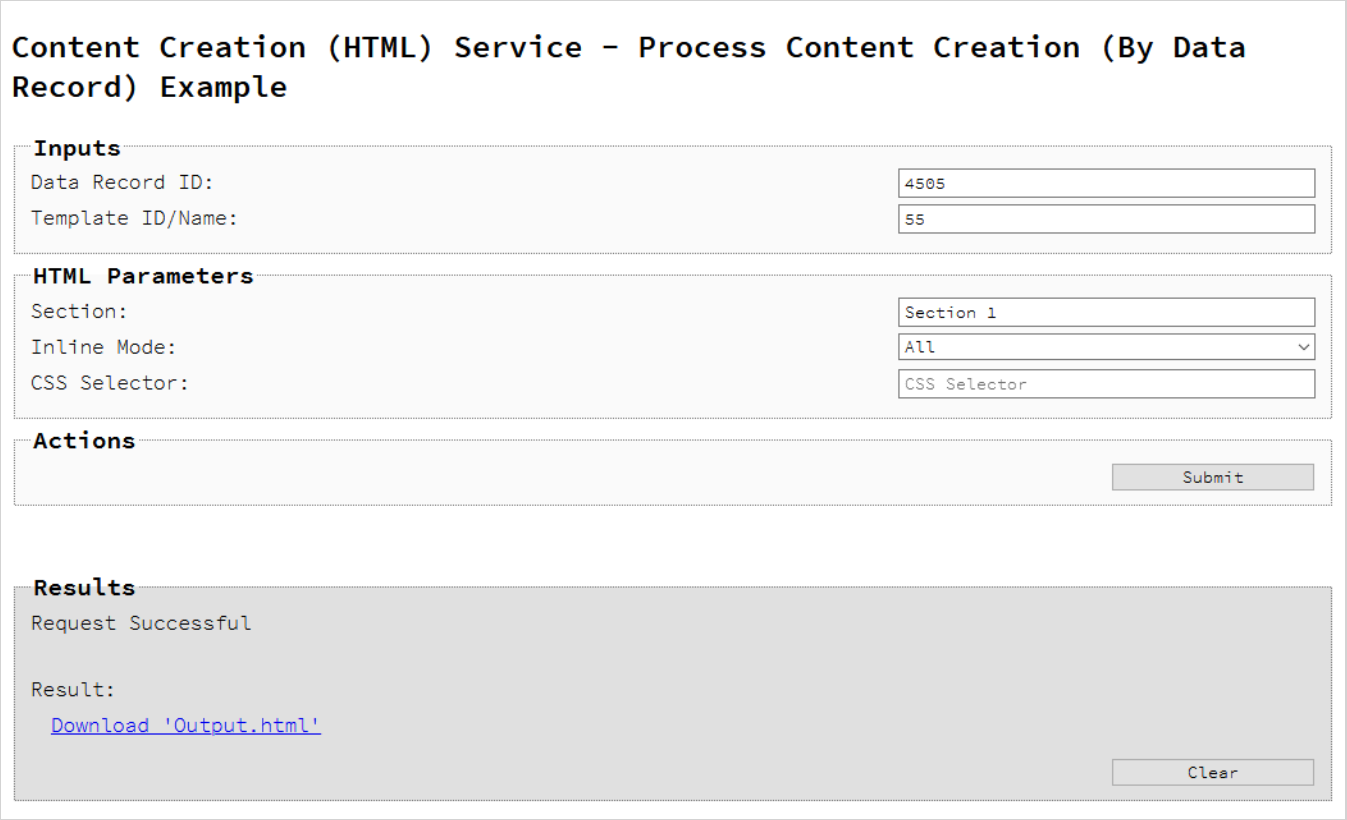
```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data Record) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cch-process-by-dre.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation (HTML) Service - Process Content Creation (By Data Record) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datarecord">Data Record ID:</label>
                    <input id="datarecord" type="text" placeholder="1234" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>HTML Parameters</legend>
                <div>
                    <label for="section">Section:</label>
                    <input id="section" type="text" placeholder="Section Name">
                </div>
                <div>
                    <label for="inline">Inline Mode:</label>
                    <select id="inline">
                        <option value="NONE">None</option>
                        <option value="CSS">CSS</option>
                        <option value="ALL">All</option>
                    </select>
                </div>
                <div>
                    <label for="cssSelector">CSS Selector:</label>
                    <input id="cssSelector" type="text" placeholder="CSS Selector">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

cch-process-by-dre.js

```
/* Content Creation (HTML) Service - Process Content Creation (By Data Record) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataRecordId    = $("#datarecord").val(),
                templateId      = $("#template").val(),
                section         = $("#section").val().trim(),
                cssSelector     = $("#cssSelector").val().trim(),
                params = {
                    inline: $("#inline").val()
                };
            if (section.length) params.section = section;
            if (cssSelector.length) params.cssSelector = cssSelector;

            /* Process Content Creation (By Data Record) */
            $.ajax({
                type:       "GET",
                url:        "/rest/serverengine/workflow/contentcreation/html/" +
                                templateId + "/" + dataRecordId,
                data:       params,
                dataType:   "file"
            })
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Result", c.dataToFileLink(response, "Output"), false);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example you first need to enter your **Data Record ID** and the Template **Managed File ID/Name** (previously uploaded to the file store) into the appropriate text fields as your inputs.

Next you can specify the HTML parameters to use when creating the web content:

- **Section** – the section within the Web context of the template to use
- **Inline Mode** – the inline mode to be used in the creation of content
- **CSS Selector** – a CSS selector for the creation of only a specific HTML element within the template

Lastly, select the **Submit** button to create and retrieve the web content. When the response returns a **Result Link** will be displayed in the **Results** area. This link can be selected to view the resulting web content that was created.

Further Reading

See the Content Creation (HTML) Service page of the REST API Reference section for further detail.

Creating Content for Web By Data Record (Using JSON)

Problem

You want to create and retrieve web content using a template and an existing Data Record as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation (HTML) REST service:

Process Content Creation (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/html/{templateId}/{dataRecordId: [0-9]+}	*POST*

Example

HTML5

cch-process-by-dre-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data Record) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cch-process-by-dre-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation (HTML) Service - Process Content Creation (By Data Record) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datarecord">Data Record ID:</label>
                    <input id="datarecord" type="text" placeholder="1234" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>HTML Parameters</legend>
                <div>
                    <label for="section">Section:</label>
                    <input id="section" type="text" placeholder="Section Name">
                </div>
                <div>
                    <label for="inline">Inline Mode:</label>
                    <select id="inline">
                        <option value="NONE">None</option>
                        <option value="CSS">CSS</option>
                        <option value="ALL">All</option>
                    </select>
                </div>
                <div>
                    <label for="cssSelector">CSS Selector:</label>
                    <input id="cssSelector" type="text" placeholder="CSS Selector">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
```

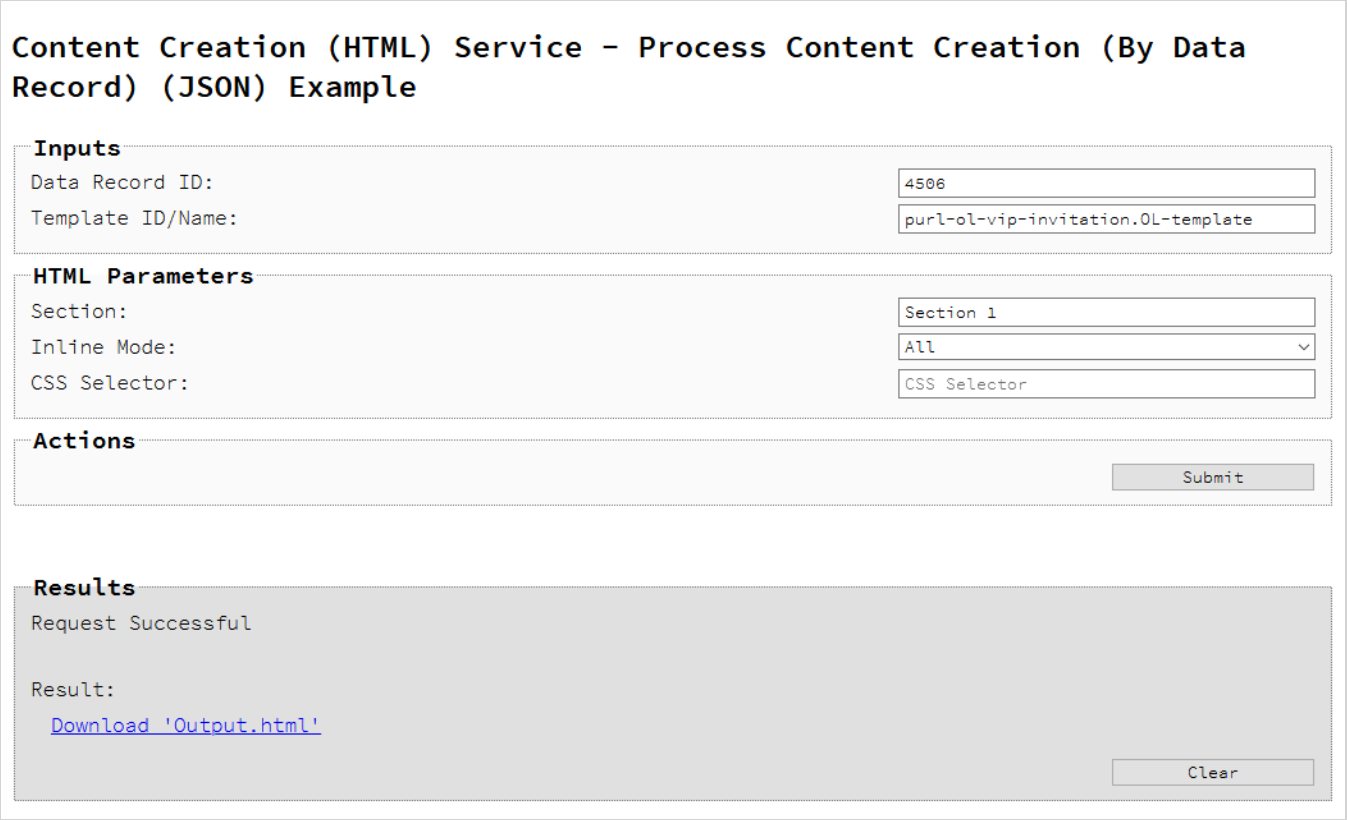
```
</html>
```

JavaScript/jQuery

cch-process-by-dre-json.js

```
/* Content Creation (HTML) Service - Process Content Creation (By Data Record) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var dataRecordId    = $("#datarecord").val(),
                templateId      = $("#template").val(),
                section         = $("#section").val().trim(),
                cssSelector     = $("#cssSelector").val().trim(),
                params = {
                    inline: $("#inline").val()
                };
            if (section.length) params.section = section;
            if (cssSelector.length) params.cssSelector = cssSelector;

            /* Process Content Creation (By Data Record) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/html/" +
                                    templateId + "/" + dataRecordId,
                data:           JSON.stringify(params),
                contentType:    "application/json",
                dataType:       "file"
            })
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Result", c.dataToFileLink(response, "Output"), false);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example you first need to enter your **Data Record ID** and the Template **Managed File ID/Name** (previously uploaded to the file store) into the appropriate text fields as your inputs.

Next you can specify the HTML parameters to use when creating the web content:

- **Section** – the section within the Web context of the template to use
- **Inline Mode** – the inline mode to be used in the creation of content
- **CSS Selector** – a CSS selector for the creation of only a specific HTML element within the template

Lastly, select the **Submit** button to create and retrieve the web content. When the response returns a **Result Link** will be displayed in the **Results** area. This link can be selected to view the resulting web content that was created.

Further Reading

See the Content Creation (HTML) Service page of the REST API Reference section for further detail.

Creating Content for Web By Data (Using JSON)

Problem

You want to create and retrieve web content using a template and JSON data as inputs.

Solution

The solution is to create a request using the following URI and method type and submit it to the server via the Content Creation (HTML) REST service:

Process Content Creation (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/html/{templateId}	*POST*

Example

HTML5

cch-process-by-data-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation (By Data) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/cch-process-by-data-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation (HTML) Service - Process Content Creation (By Data) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">JSON Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>HTML Parameters</legend>
```

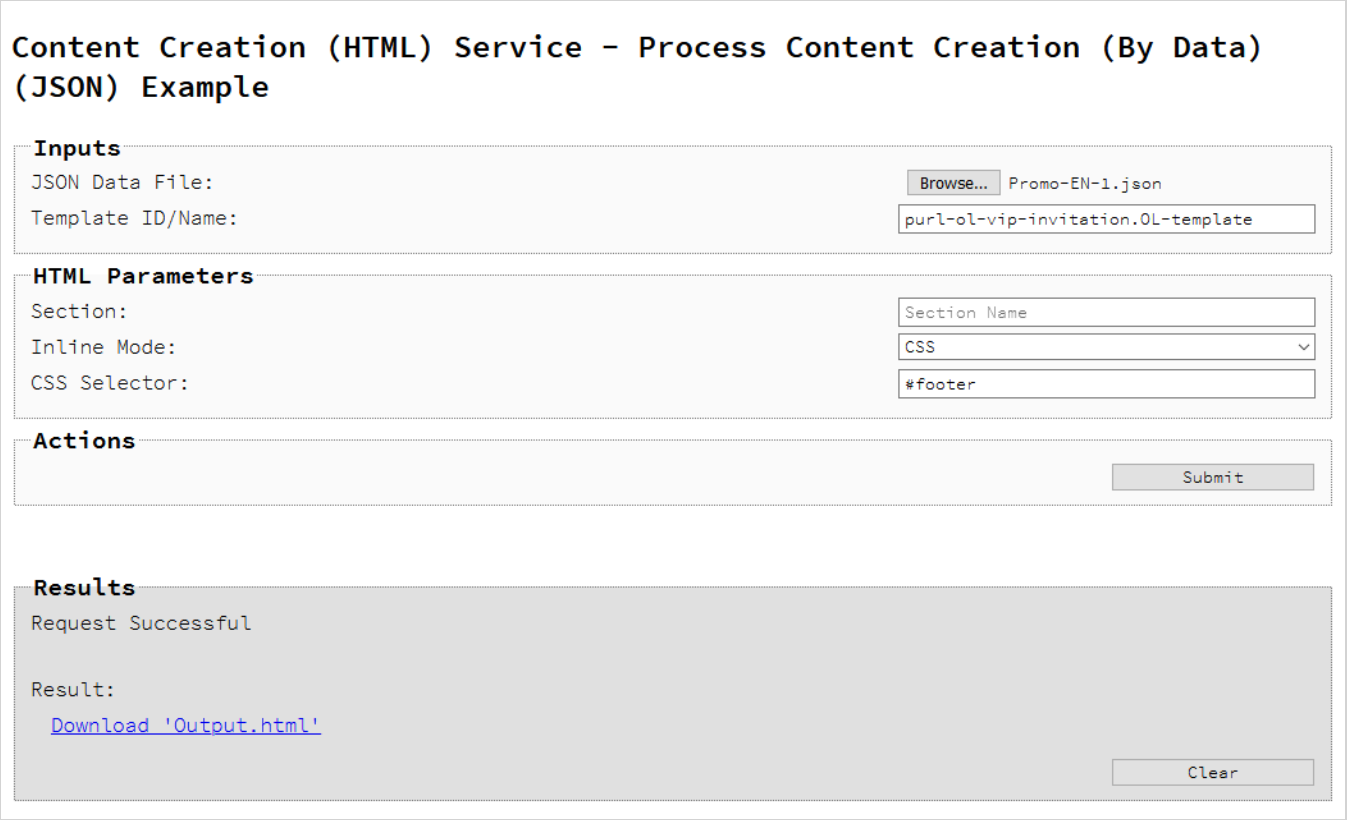
```
                <div>
                    <label for="section">Section:</label>
                    <input id="section" type="text" placeholder="Section Name">
                </div>
                <div>
                    <label for="inline">Inline Mode:</label>
                    <select id="inline">
                        <option value="NONE">None</option>
                        <option value="CSS">CSS</option>
                        <option value="ALL">All</option>
                    </select>
                </div>
                <div>
                    <label for="cssSelector">CSS Selector:</label>
                    <input id="cssSelector" type="text" placeholder="CSS Selector">
                </div>
            </fieldset>
            <fieldset>
                <legend>Actions</legend>
                <div>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

cch-process-by-data-json.js

```
/* Content Creation (HTML) Service - Process Content Creation (By Data) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var jsonData = null;

        c.setupJsonDataFileInput($("#datafile"), function (data) { jsonData = data });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var templateId  = $("#template").val(),
                section     = $("#section").val().trim(),
                inline      = $("#inline").val(),
                cssSelector = $("#cssSelector").val().trim();

            /* Process Content Creation (By Data) (JSON) */
            var settings = {
                type:           "POST",
                url:            "/rest/serverengine/workflow/contentcreation/html/" +
                                    templateId + "?inline=" + inline,
                data:           JSON.stringify({ data: jsonData }),
                contentType:    "application/json",
                dataType:       "file"
            };
            if (section.length) settings.url += "&section=" + section;
            if (cssSelector.length) settings.url += "&cssSelector=" + escape(cssSelector);

            $.ajax(settings)
                .done(function (response, status, request) {
                    c.displayStatus("Request Successful");
                    c.displayResult("Result", c.dataToFileLink(response, "Output"), false);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example you first need to use the **Browse** button to select an appropriate **JSON Data File** and then enter the **Managed File ID or Name** of your template (previously uploaded to the file store) into the appropriate text field.

Next you can specify the HTML parameters to use when creating the web content:

- **Section** – the section within the Web context of the template to use.
- **Inline Mode** – the inline mode to be used in the creation of content.
- **CSS Selector** – a CSS selector for the creation of only a specific HTML element within the template.

Lastly, select the **Submit** button to create and retrieve the web content. When the response returns a **Result Link** will be displayed in the **Results** area. This link can be selected to view the resulting web content that was created.

Further Reading

See the Content Creation (HTML) Service page of the REST API Reference section for further detail.

Running a Job Creation Operation By Content Set (Using JSON)

Problem

You want to run a job creation operation to produce a Job Set using a job creation preset and an existing set of Content Sets as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the job creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Job Creation REST service:

Process Job Creation (JSON)	/rest/serverengine/workflow/jobcreation/{configId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/jobcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/jobcreation/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/jobcreation/cancel/{operationId}	*POST*

Example

HTML5

jc-process-by-cse-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Job Creation (By Content Set) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/jc-process-by-cse-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Job Creation Service - Process Job Creation (By Content Set) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="contentsets">Content Set ID(s):</label>
                    <input id="contentsets" type="text" placeholder="1234, 2345, 3456, ..." required>
                </div>
                <div>
                    <label for="jcpreset">Job Creation Preset ID/Name:</label>
                    <input id="jcpreset" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

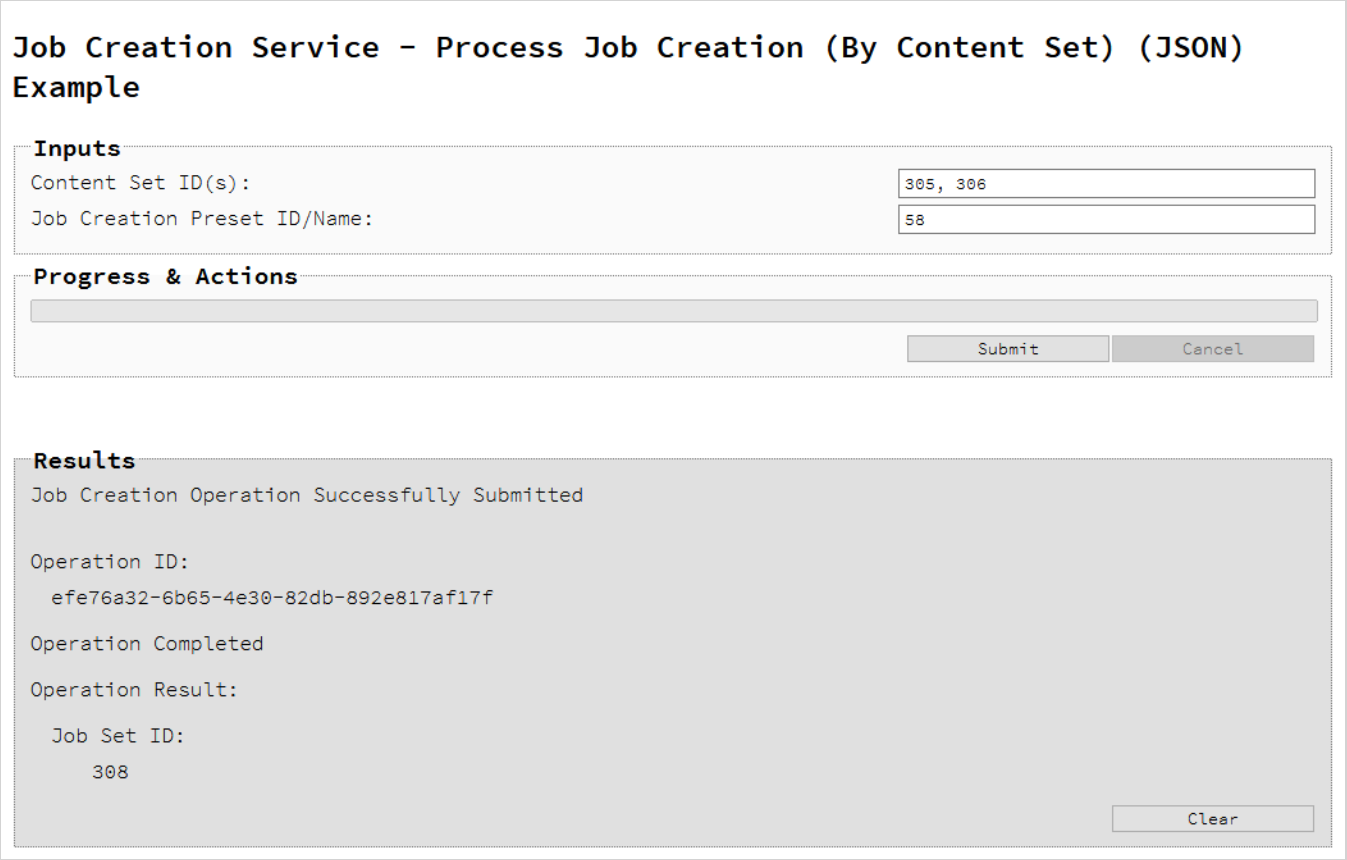
JavaScript/jQuery

jc-process-by-cse-json.js

```
/* Job Creation Service - Process Job Creation (By Content Set) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/jobcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var contentSetIds = $("#contentsets").val(),
                configId = $("#jcpreset").val();

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/jobcreation/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Job Set ID", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process Job Creation (By Content Set) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/jobcreation/" + configId,
                data:           JSON.stringify(c.plainIDListToJson(contentSetIds)),
                contentType:    "application/json"
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Job Creation Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
```

```
                    if (operationId !== null) {

                        /* Get Progress of Operation */
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/jobcreation/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter a comma delimited list of your **Content Set IDs** and the **Managed File ID or Name** of your job creation preset (previously uploaded to the file store) into the appropriate text fields, and then select the **Submit** button to start the job creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the job creation operation has completed, the ID of the Job Set created will be returned and displayed to the **Results** area.

Further Reading

See the Job Creation Service page of the REST API Reference section for further detail.

Running a Job Creation Operation By Content Set with Runtime Parameters (Using JSON)

Problem

You want to run a job creation operation to produce a Job Set using a job creation preset, an existing set of Content Sets and runtime parameters as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the job creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Job Creation REST service:

Process Job Creation (By Content Set) (Runtime Parameters) (JSON)	/rest/serverengine/workflow/jobcreation/{configId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/jobcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/jobcreation/getResult/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/jobcreation/cancel/{operationId}	*POST*

Example

HTML5

jc-process-by-cse-params-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Job Creation (By Content Set) (Runtime Parameters) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/jc-process-by-cse-params-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Job Creation Service - Process Job Creation (By Content Set) (Runtime Parameters) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="contentsets">Content Set ID(s):</label>
                    <input id="contentsets" type="text" placeholder="1234, 2345, 3456, ..." required>
                </div>
                <div>
                    <label for="jcpreset">Job Creation Preset ID/Name:</label>
                    <input id="jcpreset" type="text" placeholder="1234 or Filename" required>
                </div>
                <div>
                    <label for="parameters">Runtime Parameters:</label>
                    <table id="parameters" class="name-value parameters">
                        <tbody>
                            <tr>
                                <th></th>
                                <th>Name</th>
                                <th>Type</th>
                                <th>Value</th>
                            </tr>
```

```
                        <tr class="placeholder" hidden>
                            <td></td>
                            <td>
                                <input type="text" disabled>
                            </td>
                            <td>
                                <select disabled>
                                    <option value="string">String</option>
                                </select>
                            </td>
                            <td>
                                <input type="text" disabled>
                            </td>
                        </tr>
                    </tbody>
                </table>
            </div>
            <div>
                <input class="remove-parameter" type="button" value="Remove Selected" disabled>
                <input class="add-parameter" type="button" value="Add Parameter">
            </div>
        </fieldset>
        <fieldset>
            <legend>Progress &amp;amp; Actions</legend>
            <div>
                <progress value="0" max="100"></progress>
            </div>
            <div>
                <input id="cancel" type="button" value="Cancel" disabled>
                <input id="submit" type="submit" value="Submit">
            </div>
        </fieldset>
    </form>
</body>
</html>
```

markup.html

```
<!-- OL Connect REST API Cookbook - Working Examples [Markup HTML Snippet] -->
<div id="input-name-type-value-pair-row">
    <table>
        <tr class="root">
            <td>
                <input class="use-option" type="checkbox">
            </td>
            <td class="name">
                <input type="text" placeholder="Name" required>
            </td>
            <td class="type">
                <select id="type" class="options-selector">
                    <option value="string">String</option>
                    <option value="number">Number</option>
                    <option value="boolean">Boolean</option>
                    <option value="date">Date</option>
                </select>
            </td>
            <td class="option type-string value">
                <input type="text" placeholder="Value" required>
            </td>
            <td class="option type-number value">
                <input type="number" step="0.001" placeholder="Value" required />
            </td>
            <td class="option type-boolean value">
                <input type="checkbox" />
            </td>
            <td class="option type-date value">
                <input id="value1" type="date" required />
            </td>
        </tr>
    </table>
</div>
```

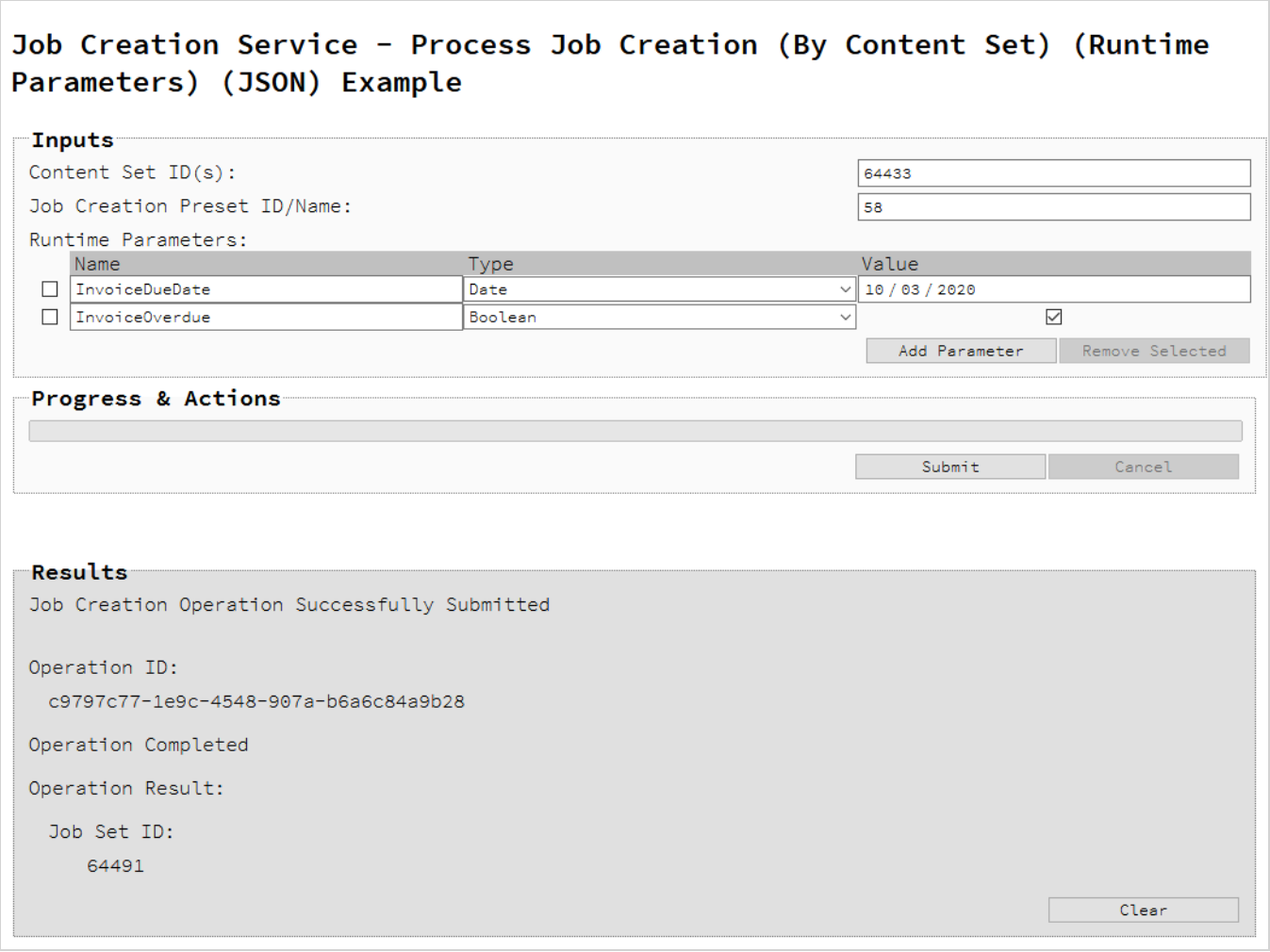
JavaScript/jQuery

jc-process-by-cse-params-json.js

```
/* Job Creation Service - Process Job Creation (By Content Set) (Runtime Parameters) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $parameters = $("#parameters"),
            $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        c.setupRuntimeParametersTableInput($parameters);

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/jobcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var contentSetIds = $("#contentsets").val(),
                configId = $("#jcpreset").val();

            var getFinalResult = function () {

                /* Get Result of Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/jobcreation/getResult/" + operationId
                })
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Job Set ID", response);
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Construct JSON Identifier List (with Runtime Parameters) */
            var jsonConfig      = c.plainIDListToJson(contentSetIds),
                jsonParameters  = c.tableToJsonRuntimeParameters($parameters);

            if (Object.keys(jsonParameters.parameters).length)
                jsonConfig.parameters = jsonParameters.parameters;

            /* Process Job Creation (By Content Set) (Runtime Parameters) (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/jobcreation/" + configId,
                data:           JSON.stringify(jsonConfig),
                contentType:    "application/json"
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Job Creation Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);
```

```
                var getProgress = function () {
                    if (operationId !== null) {

                        // Get Progress of Operation
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/jobcreation/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter a comma delimited list of your **Content Set IDs** and the **Managed File ID/Name** of your job creation preset (previously uploaded to the file store) into the appropriate text fields.

Next, specify one or more **Runtime Parameters**. Parameters can be added and removed using the **Add Parameter** and **Remove Selected** buttons respectively, and once added the following fields can be then populated for each parameter:

- **Name** – The name of the runtime parameter (as specified in the Job Creation Preset)
- **Type** – The type of the runtime parameter as either *String*, *Number*, *Boolean* or *Date* (as specified in the Job Creation Preset)
- **Value** – The value of the runtime parameter (specific to the type selected)

Lastly, select the **Submit** button to start the job creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the job creation operation has completed, the ID of the Job Set created will be returned and displayed to the **Results** area.

Further Reading

See the Job Creation Service page of the REST API Reference section for further detail.

Running an Output Creation Operation By Job Set

Problem

You want to run an output creation operation to produce print output using an output creation preset and an existing Job Set as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the output creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Output Creation REST service:

Process Output Creation	/rest/serverengine/workflow/outputcreation/{configId}/{jobSetId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/outputcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/outputcreation/getResult/{operationId}	*POST*
Get Result of Operation (as Text)	/rest/serverengine/workflow/outputcreation/getResultTxt/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/outputcreation/cancel/{operationId}	*POST*

Example

HTML5

oc-process-by-jse.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Output Creation (By Job Set) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/oc-process-by-jse.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Output Creation Service - Process Output Creation (By Job Set) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="jobset">Job Set ID:</label>
                    <input id="jobset" type="text" placeholder="1234" required>
                </div>
                <div>
                    <label for="ocpreset">Output Creation Preset ID/Name:</label>
                    <input id="ocpreset" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="resultastxt">Get Result as Text:</label>
                    <input id="resultastxt" type="checkbox">
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
```

```
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

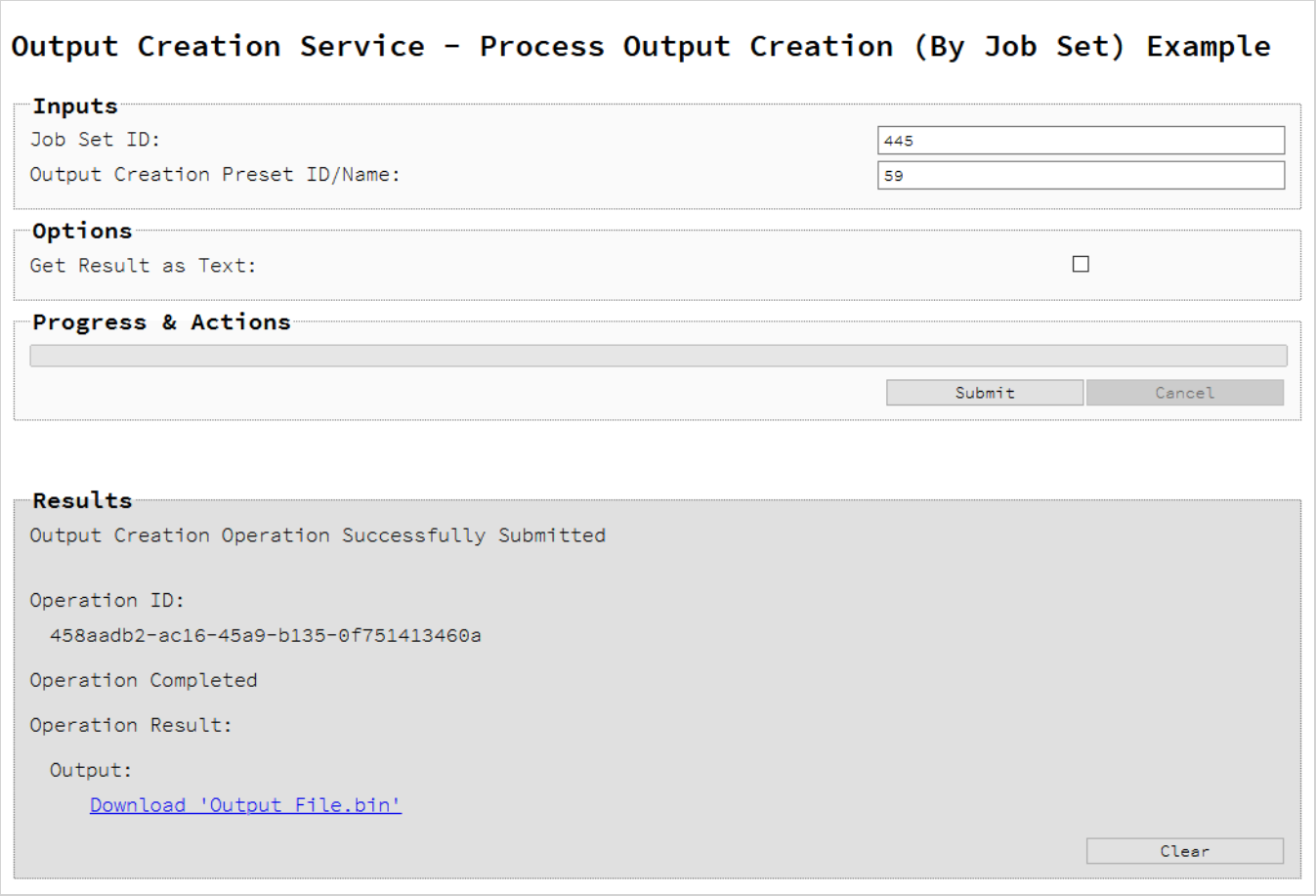
JavaScript/jQuery

oc-process-by-jse.js

```
/* Output Creation Service - Process Output Creation (By Job Set) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/outputcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var jobSetId = $("#jobset").val(),
                configId = $("#ocpreset").val();

            var getFinalResult = function () {

                var resultastxt = $("#resultastxt").prop("checked"),
                    result = (resultastxt) ? "getResultTxt" : "getResult";

                /* Get Result of Operation */
                var settings = {
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/outputcreation/" + result + "/" + operationId
                };
                if (!resultastxt) settings.dataType = "file";
                $.ajax(settings)
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        if (!resultastxt) {
                            c.displaySubResult("Output", c.dataToFileLink(response, "Output File"), false);
                        } else {
                            c.displaySubResult("Output", response);
                        }
                    })
                    .fail(c.displayDefaultFailure);
            };
```

```
            /* Process Output Creation (By Job Set) */
            $.ajax({
                type:   "POST",
                url:    "/rest/serverengine/workflow/outputcreation/" + configId + "/" + jobSetId
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("Output Creation Operation Successfully Submitted");
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
                        if (operationId !== null) {

                            /* Get Progress of Operation */
                            $.ajax({
                                type:   "GET",
                                cache:  false,
                                url:    "/rest/serverengine/workflow/outputcreation/getProgress/" + operationId
                            })
                                .done(function (response, status, request) {

                                    if (response !== "done") {
                                        if (response !== progress) {
                                            progress = response;
                                            $progressBar.attr("value", progress);
                                        }
                                        setTimeout(getProgress, 1000);
                                    } else {
                                        $progressBar.attr("value", (progress = 100));
                                        c.displayInfo("Operation Completed");
                                        getFinalResult();
                                        operationId = null;
                                        setTimeout(function () {
                                            $progressBar.attr("value", 0);
                                            $submitButton.prop("disabled", false);
                                            $cancelButton.prop("disabled", true);
                                        }, 100);
                                    }
                                })
                                .fail(c.displayDefaultFailure);
                        }
                    };
                    getProgress();
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Job Set ID** and the **Managed File ID or Name** of your output creation preset (previously uploaded to the file store) into the appropriate text fields, and then check any options that you may require:

- **Get Result as Text** – Return the result as text specifically. In this example this would return the absolute path to the output file(s).

Lastly, select the **Submit** button to start the Output creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the output creation operation has completed, the output result will be returned and displayed to the **Results** area.

Note: If the result returned is expected to be file data, then then a download link will be displayed.

Further Reading

See the Output Creation Service page of the REST API Reference section for further detail.

Running an Output Creation Operation By Job Set (Using JSON)

Problem

You want to run an output creation operation to produce print output using an output creation preset and an existing Job Set as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the output creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Output Creation REST service:

Process Output Creation (JSON)	/rest/serverengine/workflow/outputcreation/{configId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/outputcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/outputcreation/getResult/{operationId}	*POST*
Get Result of Operation (as Text)	/rest/serverengine/workflow/outputcreation/getResultTxt/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/outputcreation/cancel/{operationId}	*POST*

Example

HTML5

oc-process-by-jse-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Output Creation (By Job Set) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/oc-process-by-jse-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Output Creation Service - Process Output Creation (By Job Set) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="jobset">Job Set ID:</label>
                    <input id="jobset" type="text" placeholder="1234" required>
                </div>
                <div>
                    <label for="ocpreset">Output Creation Preset ID/Name:</label>
                    <input id="ocpreset" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="createonly">Create Only:</label>
                    <input id="createonly" type="checkbox">
                </div>
                <div>
                    <label for="resultastxt">Get Result as Text:</label>
                    <input id="resultastxt" type="checkbox">
                </div>
            </fieldset>
```

```
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

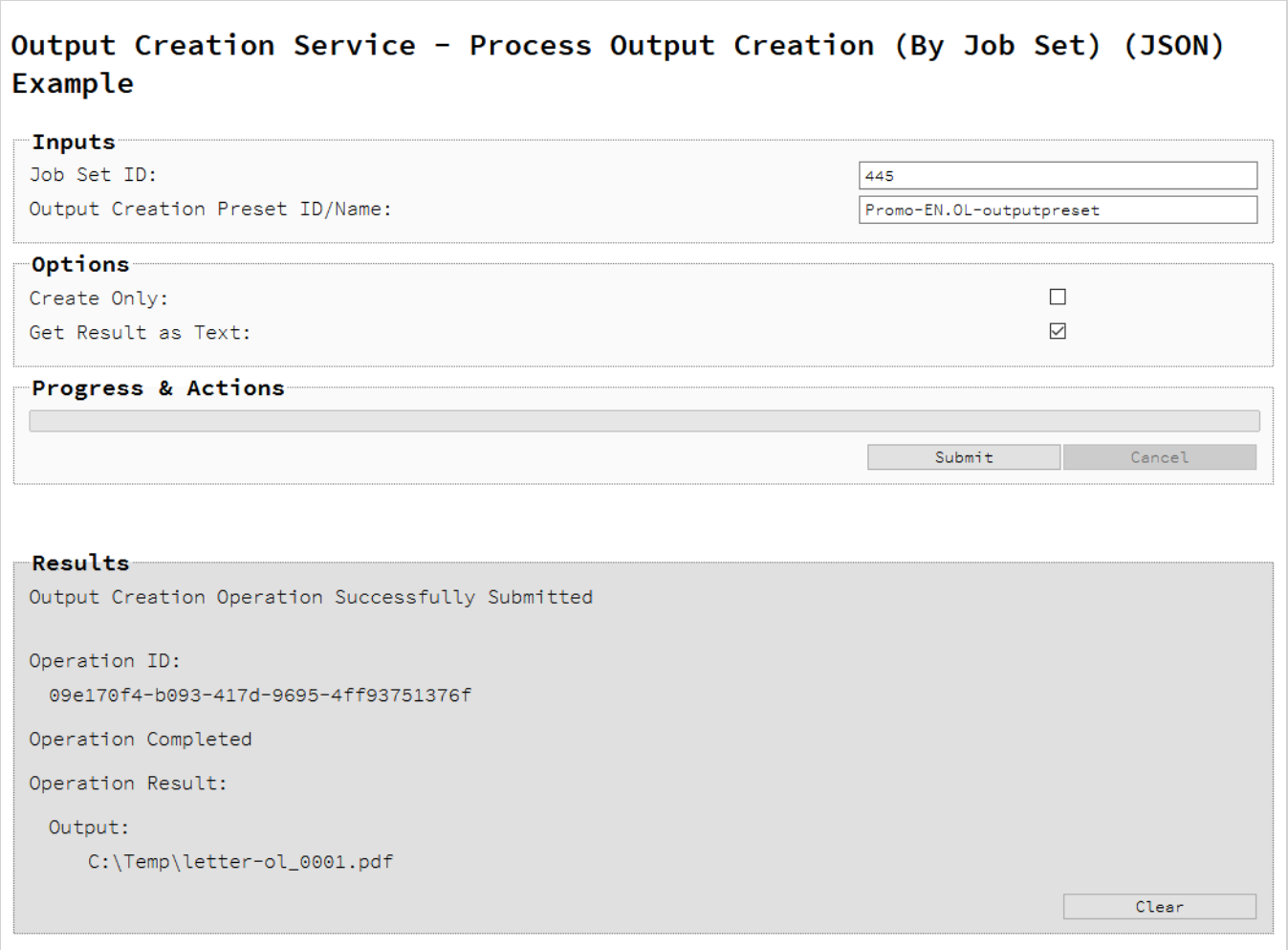
JavaScript/jQuery

oc-process-by-jse-json.js

```
/* Output Creation Service - Process Output Creation (By Job Set) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/outputcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var jobSetId = $("#jobset").val(),
                configId = $("#ocpreset").val(),
                createOnly = $("#createonly").prop("checked");

            var getFinalResult = function () {

                var resultastxt = $("#resultastxt").prop("checked"),
                    result = (resultastxt) ? "getResultTxt" : "getResult";

                /* Get Result of Operation */
                var settings = {
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/outputcreation/" + result + "/" + operationId
                };
                if (!resultastxt) settings.dataType = "file";
                $.ajax(settings)
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        if (!resultastxt) {
                            c.displaySubResult("Output", c.dataToFileLink(response, "Output File"), false);
                        } else {
```

```
                        c.displaySubResult("Output", response);
                    }
                })
                .fail(c.displayDefaultFailure);
        };

        /* Process Output Creation (By Job Set) (JSON) */
        $.ajax({
            type:           "POST",
            url:            "/rest/serverengine/workflow/outputcreation/" + configId,
            data:           JSON.stringify(c.plainIDToJson(jobSetId, createOnly)),
            contentType:    "application/json"
        })
            .done(function (response, status, request) {

                var progress = null;
                operationId = request.getResponseHeader("operationId");

                $submitButton.prop("disabled", true);
                $cancelButton.prop("disabled", false);

                c.displayStatus("Output Creation Operation Successfully Submitted");
                c.displayResult("Operation ID", operationId);

                var getProgress = function () {
                    if (operationId !== null) {

                        /* Get Progress of Operation */
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/outputcreation/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter the **Job Set ID** and the **Managed File ID or Name** of your output creation preset (previously uploaded to the file store) into the appropriate text fields, and then check any options that you may require:

- **Create Only** – Create the output in server but do not send spool file to its final destination. In this example this would mean that the output files(s) would not be sent to the output directory specified in the output creation preset.
- **Get Result as Text** – Return the result as text specifically. In this example this would return the absolute path to the output file(s).

Lastly, select the **Submit** button to start the Output creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the output creation operation has completed, the output result will be returned and displayed to the **Results** area.

> **Note:** If the result returned is expected to be file data, then then a download link will be displayed.

Further Reading

See the Output Creation Service page of the REST API Reference section for further detail.

Running an Output Creation Operation By Job (Using JSON)

Problem

You want to run an output creation operation to produce print output using an output creation preset and a list of existing Jobs as inputs.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the output creation operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the Output Creation REST service:

Process Output Creation (By Job) (JSON)	/rest/serverengine/workflow/outputcreation/{configId}/jobs	*POST*
Get Progress of Operation	/rest/serverengine/workflow/outputcreation/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/outputcreation/getResult/{operationId}	*POST*
Get Result of Operation (as Text)	/rest/serverengine/workflow/outputcreation/getResultTxt/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/outputcreation/cancel/{operationId}	*POST*

Example

HTML5

oc-process-by-je-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Output Creation (By Job) (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/oc-process-by-je-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Output Creation Service - Process Output Creation (By Job) (JSON) Example</h2>
        <form>
            <fieldset>
                <legend>Inputs</legend>
                <div>
                    <label for="jobs">Job ID(s):</label>
                    <input id="jobs" type="text" placeholder="1234, 2345, 3456, ..." required>
                </div>
                <div>
                    <label for="ocpreset">Output Creation Preset ID/Name:</label>
                    <input id="ocpreset" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="createonly">Create Only:</label>
                    <input id="createonly" type="checkbox">
                </div>
                <div>
                    <label for="resultastxt">Get Result as Text:</label>
                    <input id="resultastxt" type="checkbox">
                </div>
            </fieldset>
```

```
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

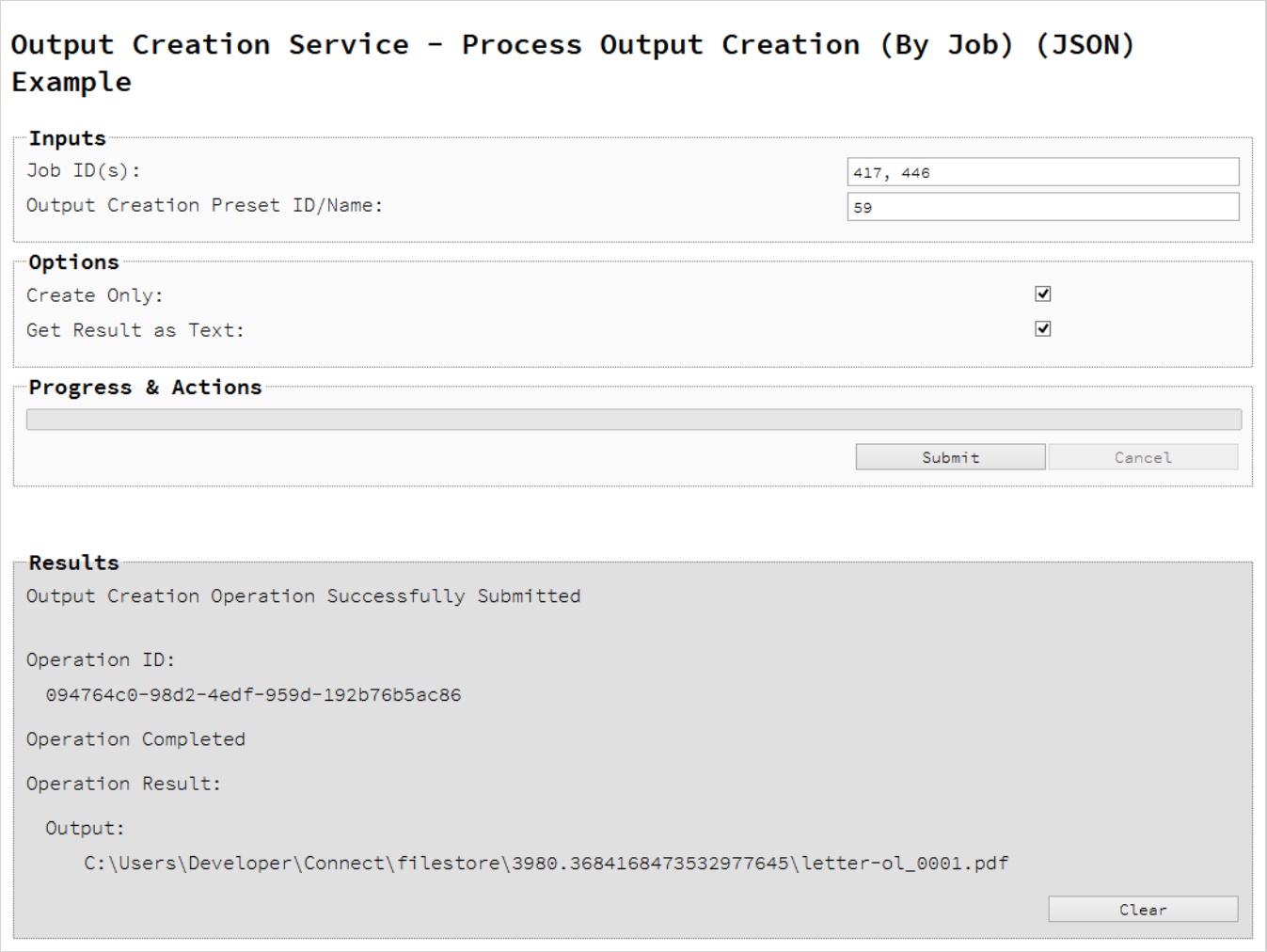
JavaScript/jQuery

oc-process-by-je-json.js

```
/* Output Creation Service - Process Output Creation (By Job) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
            if (operationId !== null) {

                /* Cancel an Operation */
                $.ajax({
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/outputcreation/cancel/" + operationId
                })
                    .done(function (response) {
                        c.displayInfo("Operation Cancelled!");
                        operationId = null;
                        setTimeout(function () {
                            $progressBar.attr("value", 0);
                            $submitButton.prop("disabled", false);
                            $cancelButton.prop("disabled", true);
                        }, 100);
                    })
                    .fail(c.displayDefaultFailure);
            }
        });

        $("form").on("submit", function (event) {

            event.preventDefault();
            if (!c.checkSessionValid()) return;

            var jobIds = $("#jobs").val(),
                configId = $("#ocpreset").val(),
                createOnly = $("#createonly").prop("checked");

            var getFinalResult = function () {

                var resultastxt = $("#resultastxt").prop("checked"),
                    result = (resultastxt) ? "getResultTxt" : "getResult";

                /* Get Result of Operation */
                var settings = {
                    type:   "POST",
                    url:    "/rest/serverengine/workflow/outputcreation/" + result + "/" + operationId
                };
                if (!resultastxt) settings.dataType = "file";
                $.ajax(settings)
                    .done(function (response, status, request) {
                        c.displayHeading("Operation Result");
                        if (!resultastxt) {
                            c.displaySubResult("Output", c.dataToFileLink(response, "Output File"), false);
                        } else {
```

```
                        c.displaySubResult("Output", response);
                    }
                })
                .fail(c.displayDefaultFailure);
        };

        /* Process Output Creation (By Job) (JSON) */
        $.ajax({
            type:           "POST",
            url:            "/rest/serverengine/workflow/outputcreation/" + configId + "/jobs",
            data:           JSON.stringify(c.plainIDListToJson(jobIds, createOnly)),
            contentType:    "application/json"
        })
            .done(function (response, status, request) {

                var progress = null;
                operationId = request.getResponseHeader("operationId");

                $submitButton.prop("disabled", true);
                $cancelButton.prop("disabled", false);

                c.displayStatus("Output Creation Operation Successfully Submitted");
                c.displayResult("Operation ID", operationId);

                var getProgress = function () {
                    if (operationId !== null) {

                        /* Get Progress of Operation */
                        $.ajax({
                            type:   "GET",
                            cache:  false,
                            url:    "/rest/serverengine/workflow/outputcreation/getProgress/" + operationId
                        })
                            .done(function (response, status, request) {

                                if (response !== "done") {
                                    if (response !== progress) {
                                        progress = response;
                                        $progressBar.attr("value", progress);
                                    }
                                    setTimeout(getProgress, 1000);
                                } else {
                                    $progressBar.attr("value", (progress = 100));
                                    c.displayInfo("Operation Completed");
                                    getFinalResult();
                                    operationId = null;
                                    setTimeout(function () {
                                        $progressBar.attr("value", 0);
                                        $submitButton.prop("disabled", false);
                                        $cancelButton.prop("disabled", true);
                                    }, 100);
                                }
                            })
                            .fail(c.displayDefaultFailure);
                    }
                };
                getProgress();
            })
            .fail(c.displayDefaultFailure);
    });
});
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example simply enter a comma delimited list of your **Job IDs** and the **Managed File ID or Name** of your output creation preset (previously uploaded to the file store) into the appropriate text fields, and then check any options that you may require:

- **Create Only** – Create the output in server but do not send spool file to its final destination. In this example this would mean that the output files(s) would not be sent to the output directory specified in the output creation preset.
- **Get Result as Text** – Return the result as text specifically. In this example this would return the absolute path to the output file(s).

Lastly, select the **Submit** button to start the Output creation operation.

Once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the output creation operation has completed, the output result will be returned and displayed to the **Results** area.

> **Note:** If the result returned is expected to be file data, then then a download link will be displayed.

Further Reading

See the Output Creation Service page of the REST API Reference section for further detail.

Running an All-In-One Operation (Using JSON)

Problem

You want to run an All-In-One operation to produce either a Data Set, Content Sets, a Job Set or print output using one of the available process and input combinations.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the All-In-One operation. There is also the option of cancelling an operation during processing if required. These requests can be submitted via the All-In-One REST service:

Process All-In-One (JSON)	/rest/serverengine/workflow/print/submit	*POST*
Get Progress of Operation	/rest/serverengine/workflow/print/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/print/getResult/{operationId}	*POST*
Get Result of Operation (as Text)	/rest/serverengine/workflow/print/getResultTxt/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/print/cancel/{operationId}	*POST*

Example

HTML5

aio-process-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process All-In-One (JSON) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/aio-process-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>All-In-One Service - Process All-In-One (JSON) Example</h2>
        <form>
            <fieldset id="inputs">
                <legend>Inputs</legend>
                <div>
                    <label for="datamining">Data Mapping:</label>
                    <input id="datamining" type="checkbox">
                </div>
                <div>
                    <label for="contentcreation">Content Creation:</label>
                    <input id="contentcreation" type="checkbox">
                </div>
                <div>
                    <label for="jobcreation">Job Creation:</label>
                    <input id="jobcreation" type="checkbox">
                </div>
                <div>
                    <label for="outputcreation">Output Creation:</label>
                    <input id="outputcreation" type="checkbox">
                </div>
            </fieldset>
            <fieldset id="datamining-inputs" disabled>
                <legend>Data Mapping</legend>
                <div>
```

```
            <label for="datafile">Data File ID/Name:</label>
            <input id="datafile" type="text" placeholder="1234 or Filename" required>
        </div>
        <div>
            <label for="datamapper">Data Mapping Configuration ID/Name:</label>
            <input id="datamapper" type="text" placeholder="1234 or Filename" required>
        </div>
    </fieldset>
    <fieldset id="contentcreation-inputs" disabled>
        <legend>Content Creation</legend>
        <div>
            <label for="datarecords">Data Record ID(s):</label>
            <input id="datarecords" type="text" placeholder="1234, 2345, 3456, ..." required>
        </div>
        <div>
            <label for="template">Template ID/Name:</label>
            <input id="template" type="text" placeholder="1234 or Filename" required>
        </div>
    </fieldset>
    <fieldset id="jobcreation-inputs" disabled>
        <legend>Job Creation</legend>
        <div>
            <label for="jcpreset">Job Creation Preset ID/Name:</label>
            <input id="jcpreset" type="text" placeholder="1234 or Filename" disabled>
        </div>
        <div>
            <label for="parameters">Runtime Parameters:</label>
            <table id="parameters" class="name-value parameters">
                <tbody>
                    <tr>
                        <th></th>
                        <th>Name</th>
                        <th>Type</th>
                        <th>Value</th>
                    </tr>
                    <tr class="placeholder" hidden>
                        <td></td>
                        <td>
                            <input type="text" disabled>
                        </td>
                        <td>
                            <select disabled>
                                <option value="string">String</option>
                            </select>
                        </td>
                        <td>
                            <input type="text" disabled>
                        </td>
                    </tr>
                </tbody>
            </table>
        </div>
        <div>
            <input class="remove-parameter" type="button" value="Remove Selected" disabled>
            <input class="add-parameter" type="button" value="Add Parameter">
        </div>
    </fieldset>
    <fieldset id="outputcreation-inputs" disabled>
        <legend>Output Creation</legend>
        <div>
            <label for="jobs">Job ID(s):</label>
            <input id="jobs" type="text" placeholder="1234, 2345, 3456, ..." required>
        </div>
        <div>
            <label for="ocpreset">Output Creation Preset ID/Name:</label>
            <input id="ocpreset" type="text" placeholder="1234 or Filename" required>
        </div>
    </fieldset>
    <fieldset>
        <legend>Options</legend>
        <div>
            <label for="persistdres">Persist Data Records:</label>
            <input id="persistdres" type="checkbox" disabled checked>
        </div>
        <div>
            <label for="createonly">Create Only:</label>
            <input id="createonly" type="checkbox" disabled>
        </div>
        <div>
            <label for="resultastxt">Get Result as Text:</label>
            <input id="resultastxt" type="checkbox" disabled>
        </div>
```

```
                <div>
                    <label for="printrange">Print Range:</label>
                    <input id="printrange" type="text" placeholder="1, 2, 3-5, 6" disabled>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

markup.html

```
<!-- OL Connect REST API Cookbook - Working Examples [Markup HTML Snippet] -->
<div id="input-name-type-value-pair-row">
    <table>
        <tr class="root">
            <td>
                <input class="use-option" type="checkbox">
            </td>
            <td class="name">
                <input type="text" placeholder="Name" required>
            </td>
            <td class="type">
                <select id="type" class="options-selector">
                    <option value="string">String</option>
                    <option value="number">Number</option>
                    <option value="boolean">Boolean</option>
                    <option value="date">Date</option>
                </select>
            </td>
            <td class="option type-string value">
                <input type="text" placeholder="Value" required>
            </td>
            <td class="option type-number value">
                <input type="number" step="0.001" placeholder="Value" required />
            </td>
            <td class="option type-boolean value">
                <input type="checkbox" />
            </td>
            <td class="option type-date value">
                <input id="value1" type="date" required />
            </td>
        </tr>
    </table>
</div>
```

JavaScript/jQuery

aio-process-json.js

```
/* All-In-One Service - Process All-In-One (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $form =     $("form"),
            $inputs =   $("#inputs input"),

            $datafile =     $("#datafile"),
```

```
    $datamapper =   $("#datamapper"),
    $datarecords =  $("#datarecords"),
    $template =     $("#template"),
    $parameters =   $("#parameters"),
    $jcpreset =     $("#jcpreset"),
    $jobs =         $("#jobs"),
    $ocpreset =     $("#ocpreset"),
    $persistdres =  $("#persistdres"),
    $createonly =   $("#createonly"),
    $resultastxt =  $("#resultastxt"),
    $printrange =   $("#printrange"),

    AIOConfig =     null,
    outputDesc =    null,
    operationId =   null,

    $submitButton = $("#submit"),
    $cancelButton = $("#cancel"),
    $progressBar =  $("progress");

c.setupRuntimeParametersTableInput($parameters);

$printrange
    .on("change", function (event) {
        c.setCustomInputValidity(event.target, c.validateNumericRange,
            "Invalid Print Range value entered");
    })
    .trigger("change");

$cancelButton.on("click", function () {
    if (operationId !== null) {

        /* Cancel an Operation */
        $.ajax({
            type:   "POST",
            url:    "/rest/serverengine/workflow/print/cancel/" + operationId
        })
            .done(function (response) {
                c.displayInfo("Operation Cancelled!");
                operationId = null;
                setTimeout(function () {
                    $progressBar.attr("value", 0);
                    $submitButton.prop("disabled", false);
                    $cancelButton.prop("disabled", true);
                }, 100);
            })
            .fail(c.displayDefaultFailure);
    }
});

/**
 * @function generateAIOConfig
 * @description Validates the workflow selected by the user
 * and constructs and an All-In-One Configuration using the relevant
 * input fields in the HTML Form.
 * Any invalid inputs or workflow selections will be red-flagged in
 * the HTML Form. Null can also be returned if no workflow selections
 * are made or if the workflow selections made are of an invalid sequence.
 * @private
 * @returns {Object} The All-In-One Configuration Object or Null
 */
function generateAIOConfig() {

    var config = {},
        required = [],
        i = null,

        /* Parse Input Value to JSON Identifier List (Helper Function) */
        jsonIDListValue = function ($input) {
            return (c.plainIDListToJson($input.val())).identifiers;
        },

        /* Parse Input Value to JSON Runtime Parameters (Helper Function) */
        jsonRuntimeParametersValue = function ($input) {
            return (c.tableToJsonRuntimeParameters($input)).parameters;
        },

        /* Parse Input Value to Boolean (Helper Function) */
        booleanValue = function ($input) {
            return $input.prop("checked");
        };
```

```
/* Get Input Value and add it to the Configuration (Helper Function) */
function getInputValue($input, process, field, parser) {
    var value = $input.val().trim();
    if (value) {
        if (parser)
            value = parser($input);
        if (config[process] === undefined)
            config[process] = {};
        config[process][field] = value;
    }
}

/* Get Table Input Value and add it to the Configuration (Helper Function) */
function getTableInputValue($table, process, field, parser) {
    if ($table.find("tr:has(td)").not(".placeholder").length) {
        if (config[process] === undefined)
            config[process] = {};
        config[process][field] = parser($table);
    }
}

/* Get Required & Actual Workflow Selections */
$inputs.each(function () {
    if ($(this).prop("checked"))
        config[this.id] = {};
    $(this).prop("required", false);
    required.push(this.id);
});
var selections = (Object.keys(config)).length;

/* Verify the Workflow Selections and note any omissions */
var matches = 0,
    missing = [];
for (i = 0; i < required.length; i += 1) {
    var step = required[i];
    if (config[step]) {
        if (!matches && step === "jobcreation")
            missing.push("contentcreation");
        matches += 1;
    } else {
        if (matches !== 0) missing.push(step);
    }
    if (matches === selections) break;
}

/* Add the inputs to the Workflow Selections to Create the All-In-One Configuration */
if (config.datamining) {
    getInputValue($datafile, "datamining", "identifier");
    getInputValue($datamapper, "datamining", "config");
    outputDesc = "Data Set ID";
}
if (config.contentcreation) {
    getInputValue($template, "contentcreation", "config");
    if (!config.datamining) {
        getInputValue($datarecords, "contentcreation", "identifiers", jsonIDListValue);
        $datarecords.prop("disabled", false);
    } else {
        $datarecords.prop("disabled", true);
    }
    outputDesc = "Content Set ID(s)";
}
if (config.jobcreation) {
    getInputValue($jcpreset, "jobcreation", "config");
    getTableInputValue($parameters, "jobcreation", "parameters",
        jsonRuntimeParametersValue);
    $jcpreset.prop("disabled", false);
    outputDesc = "Job Set ID";
} else {
    $jcpreset.prop("disabled", true);
}
if (config.outputcreation) {
    getInputValue($ocpreset, "outputcreation", "config");
    getInputValue($createonly, "outputcreation", "createOnly", booleanValue);
    if (!config.contentcreation) {
        getInputValue($jobs, "outputcreation", "identifiers", jsonIDListValue);
        $jobs.prop("disabled", false);
    } else {
        $jobs.prop("disabled", true);
    }
    $createonly.prop("disabled", false);
    $resultastxt.prop("disabled", false);
    outputDesc = "Output";
```

```
    } else {
        $createonly.prop("disabled", true);
        $resultastxt.prop({ "disabled": true, "checked": true });
    }

    if (config.datamining) {
        if (config.jobcreation && config.jobcreation.config) {
            $persistdres.prop({ "disabled": true, "checked": true });
        } else {
            getInputValue($persistdres, "datamining", "persistDataset", booleanValue);
            $persistdres.prop("disabled", false);
        }
    } else {
        $persistdres.prop({ "disabled": true });
    }

    if (config.datamining && config.contentcreation &&
            config.jobcreation && config.outputcreation) {
        getInputValue($printrange, "printRange", "printRange");
        $printrange.prop("disabled", false);
    } else {
        $printrange.prop("disabled", true);
    }

    if (config.jobcreation && !config.jobcreation.config &&
            config.jobcreation.parameters) {
        $jcpreset.prop("required", true);
    } else {
        $jcpreset.prop("required", false);
    }

    /* Red-flag any omissions in Workflow Selections */
    if (!selections || missing.length) {
        for (i = 0; i < missing.length; i += 1)
            $("#" + missing[i]).prop("required", true);
        return null;
    }
    return config;
}

$inputs
    .on("change", function (event) {
        var input = event.target;
        var process = $("#" + input.id + "-inputs");
        process.prop("disabled", !($(input).prop("checked")));
    })
    .trigger("change");

$form
    .on("change", function (event) {
        AIOConfig = generateAIOConfig();
    })
    .on("submit", function (event) {

        event.preventDefault();
        if (!c.checkSessionValid()) return;

        if (!AIOConfig) {
            alert("Invalid All-In-One Configuration!\n\nPlease enter a valid " +
                "combination of input fields, and try again.");
            return;
        }

        var getFinalResult = function () {

            var resultastxt = $resultastxt.prop("checked"),
                result = (resultastxt) ? "getResultTxt" : "getResult";

            /* Get Result of Operation */
            var settings = {
                type:   "POST",
                url:    "/rest/serverengine/workflow/print/" + result + "/" + operationId
            };
            if (!resultastxt) settings.dataType = "file";
            $.ajax(settings)
                .done(function (response, status, request) {
                    c.displayHeading("Operation Result");
                    if (!resultastxt) {
                        c.displaySubResult(outputDesc, c.dataToFileLink(response, "Output File"), false);
                    } else {
                        c.displaySubResult(outputDesc, response);
                    }
```

```
                    })
                    .fail(c.displayDefaultFailure);
            };

            /* Process All-In-One (JSON) */
            $.ajax({
                type:           "POST",
                url:            "/rest/serverengine/workflow/print/submit",
                data:           JSON.stringify(AIOConfig),
                contentType:    "application/json"
            })
                .done(function (response, status, request) {

                    var progress = null;
                    operationId = request.getResponseHeader("operationId");

                    $submitButton.prop("disabled", true);
                    $cancelButton.prop("disabled", false);

                    c.displayStatus("All-In-One Operation Successfully Submitted");
                    c.displayHeading("Input Configuration");
                    c.displaySubResult("JSON All-In-One Configuration", c.jsonPrettyPrint(AIOConfig));
                    c.displayResult("Operation ID", operationId);

                    var getProgress = function () {
                        if (operationId !== null) {

                            /* Get Progress of Operation */
                            $.ajax({
                                type:   "GET",
                                cache:  false,
                                url:    "/rest/serverengine/workflow/print/getProgress/" + operationId
                            })
                                .done(function (response, status, request) {

                                    if (response !== "done") {
                                        if (response !== progress) {
                                            progress = response;
                                            $progressBar.attr("value", progress);
                                        }
                                        setTimeout(getProgress, 1000);
                                    } else {
                                        $progressBar.attr("value", (progress = 100));
                                        c.displayInfo("Operation Completed");
                                        getFinalResult();
                                        operationId = null;
                                        setTimeout(function () {
                                            $progressBar.attr("value", 0);
                                            $submitButton.prop("disabled", false);
                                            $cancelButton.prop("disabled", true);
                                        }, 100);
                                    }
                                })
                                .fail(c.displayDefaultFailure);
                        }
                    };
                    getProgress();
                })
                .fail(c.displayDefaultFailure);
        })
        .trigger("change");
});
}(jQuery, Common));
```

Screenshot & Output

All-In-One Service - Process All-In-One (JSON) Example

Inputs

Data Mapping: ☑
Content Creation: ☑
Job Creation: ☑
Output Creation: ☑

Data Mapping

Data File ID/Name: Promo-EN-1000.csv
Data Mapping Configuration ID/Name: Promo-EN.OL-datamapper

Content Creation

Data Record ID(s): 4505, 4506, 4507, 4508
Template ID/Name: letter-ol.OL-template

Job Creation

Job Creation Preset ID/Name: 58

Runtime Parameters:

	Name	Type	Value
☐	InvoiceDueDate	Date	10 / 03 / 2020
☐	InvoiceOverdue	Boolean	☑

Add Parameter | Remove Selected

Output Creation

Job ID(s): 417, 446
Output Creation Preset ID/Name: 59

Options

Persist Data Records: ☑
Create Only: ☐
Get Result as Text: ☑
Print Range: 1-5, 7, 9

Progress & Actions

Submit | Cancel

Results

All-In-One Operation Successfully Submitted

Input Configuration:

JSON All-In-One Configuration:


```
{
    "datamining": {
        "identifier": "Promo-EN-1000.csv",
        "config": "Promo-EN.OL-datamapper"
    },
    "contentcreation": {
        "config": "letter-ol.OL-template"
    },
    "jobcreation": {
        "config": "58",
        "parameters": {
            "InvoiceDueDate": "2020-03-10",
            "InvoiceOverdue": true
        }
```

Usage

To run the example simply select the input combination of your choosing, populate the appropriate input fields and then check any options that you may require.

The following file based input fields can be referenced by **Managed File ID\Name**:

- Data file
- Data Mapping configuration
- Template
- Job Creation preset
- Output Creation preset

Specific to the specification of a Job Creation Preset, one or more **Runtime Parameters** can also be specified. Parameters can be added and removed using the **Add Parameter** and **Remove Selected** buttons respectively, and once added the following fields can be then populated for each parameter:

- **Name** – The name of the runtime parameter (as specified in the Job Creation Preset)
- **Type** – The type of the runtime parameter as either *String*, *Number*, *Boolean* or *Date* (as specified in the Job Creation Preset)
- **Value** – The value of the runtime parameter (specific to the type selected)

The following options are only available if the input combination includes output creation:

- **Create Only** – Create the output in server but do not send spool file to its final destination. In this example this would mean that the output files(s) would not be sent to the output directory specified in the output creation preset.
- **Get Result as Text** – Return the result as text specifically. If our All-In-One Configuration includes output creation, then in this example this would return the absolute path to the output file (s).

The following option is only available if the input combination includes *all* four processes:

- **Print Range** – Restrict the print output to a specific range of records in the input data, not a specific range of pages.

The following option is only available if the input combination specifically includes the data mapping process but with no job creation preset specified:

- **Persist Data Records** – Create/persist data records entities in the server during the data mapping process (intended for use with once-off jobs where the storage of data records in the server

is not required).

Lastly, select the **Submit** button to start the All-In-One operation.

Once the operation has started processing, the JSON All-In-One Configuration along with the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the All-in-One operation has completed, the result will be returned and displayed to the **Results** area.

If the All-In-One configuration includes output creation, then the result returned will be the output files (either their absolute path(s) or the output file itself). If the configuration does not include output creation, then the result returned will be either a Data Set ID, Content Set IDs or Job Set ID.

> **Note:** If the result returned is expected to be file data, then then a download link will be displayed.

Further Reading

See the All-In-One Service page of the REST API Reference section for further detail.

Running an All-In-One Operation with Adhoc Data

Problem

You want to run an All-In-One operation to produce print output from an ad hoc data file.

Solution

The solution is to make a series of requests using the following URIs and method types to submit, monitor progress and ultimately retrieve the result of the All-In-One operation. These requests can be submitted via the All-In-One REST service:

Process All-In-One (Adhoc Data)	/rest/serverengine/workflow/print/{dmConfigId}/{templateId}/{jcConfigId}/{ocConfigId}	*POST*
Get Progress of Operation	/rest/serverengine/workflow/print/getProgress/{operationId}	*GET*
Get Result of Operation	/rest/serverengine/workflow/print/getResult/{operationId}	*POST*
Get Result of Operation (as Text)	/rest/serverengine/workflow/print/getResultTxt/{operationId}	*POST*
Cancel an Operation	/rest/serverengine/workflow/print/cancel/{operationId}	*POST*

Example

HTML5

aio-process-adhoc-data.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process All-In-One (Adhoc Data) Example</title>
        <script src="../../common/lib/js/jquery-3.6.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/aio-process-adhoc-data.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>All-In-One Service - Process All-In-One (Adhoc Data) Example</h2>
        <form>
            <fieldset id="inputs">
                <legend>Inputs</legend>
                <div>
                    <label for="datafile">Data File:</label>
                    <input id="datafile" type="file" required>
                </div>
                <div>
                    <label for="datamapper">Data Mapping Configuration ID/Name:</label>
                    <input id="datamapper" type="text" placeholder="1234 or Filename" required>
                </div>
                <div>
                    <label for="template">Template ID/Name:</label>
                    <input id="template" type="text" placeholder="1234 or Filename" required>
                </div>
                <div>
                    <label for="jcpreset">Job Creation Preset ID/Name:</label>
                    <input id="jcpreset" type="text" placeholder="1234 or Filename">
                </div>
                <div>
                    <label for="ocpreset">Output Creation Preset ID/Name:</label>
                    <input id="ocpreset" type="text" placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Options</legend>
                <div>
                    <label for="async">Asynchronous:</label>
                    <input id="async" type="checkbox" checked>
                </div>
                <div>
                    <label for="createonly">Create Only:</label>
                    <input id="createonly" type="checkbox">
                </div>
                <div>
                    <label for="resultastxt">Get Result as Text:</label>
                    <input id="resultastxt" type="checkbox">
                </div>
                <div>
                    <label for="printrange">Print Range:</label>
                    <input id="printrange" type="text" placeholder="1, 2, 3-5, 6">
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &amp;amp; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
                <div>
                    <input id="cancel" type="button" value="Cancel" disabled>
                    <input id="submit" type="submit" value="Submit">
                </div>
            </fieldset>
        </form>
    </body>
</html>
```

JavaScript/jQuery

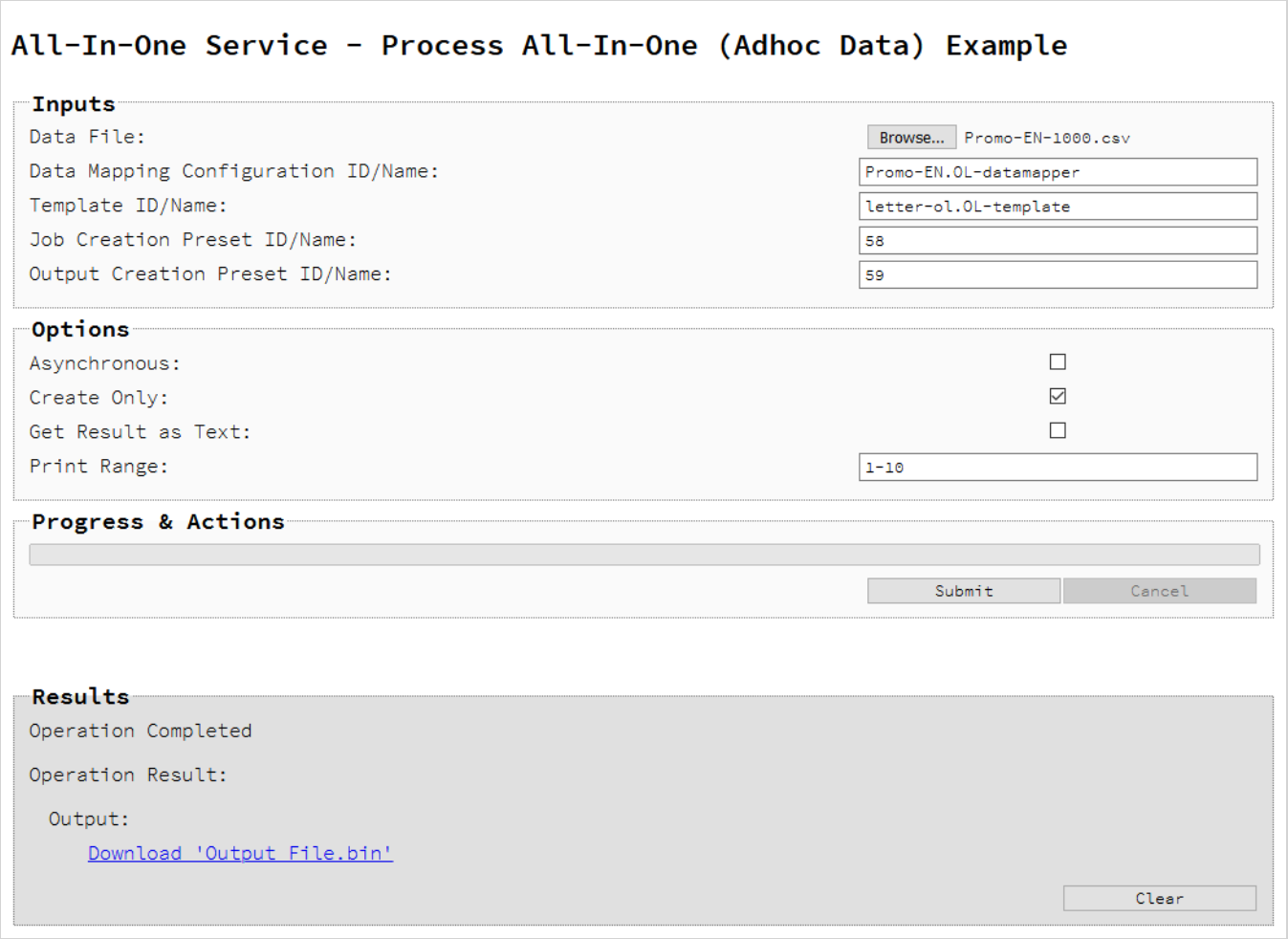
aio-process-adhoc-data.js

```
/* All-In-One Service - Process All-In-One (Adhoc Data) Example */
(function ($, c) {
    "use strict";
    $(function () {
```

```
c.setupExample();

var $submitButton = $("#submit"),
    $cancelButton = $("#cancel"),
    $progressBar =  $("progress"),
    $printrange = $("#printrange"),
    operationId =   null;

$printrange
    .on("change", function (event) {
        c.setCustomInputValidity(event.target, c.validateNumericRange,
            "Invalid Print Range value entered");
    })
    .trigger("change");

$cancelButton.on("click", function () {
    if (operationId !== null) {

        /* Cancel an Operation */
        $.ajax({
            type:   "POST",
            url:    "/rest/serverengine/workflow/print/cancel/" + operationId
        })
            .done(function (response) {
                c.displayInfo("Operation Cancelled!");
                operationId = null;
                setTimeout(function () {
                    $progressBar.attr("value", 0);
                    $submitButton.prop("disabled", false);
                    $cancelButton.prop("disabled", true);
                }, 100);
            })
            .fail(c.displayDefaultFailure);
    }
});

$("form").on("submit", function (event) {

    event.preventDefault();
    if (!c.checkSessionValid()) return;

    var file = $("#datafile")[0].files[0],

        dmConfigId = $("#datamapper").val(),
        templateId = $("#template").val(),
        jcConfigId = $("#jcpreset").val() || "0",
        ocConfigId = $("#ocpreset").val(),

        async = $("#async").prop("checked"),
        createOnly = $("#createonly").prop("checked"),
        resultAsTxt = $("#resultastxt").prop("checked"),
        printRange = $printrange.val();

    var getFinalResult = function () {

        var result = (resultAsTxt) ? "getResultTxt" : "getResult";

        /* Get Result of Operation */
        var settings = {
            type:   "POST",
            url:    "/rest/serverengine/workflow/print/" + result + "/" + operationId
        };
        if (!resultAsTxt) settings.dataType = "file";
        $.ajax(settings)
            .done(function (response, status, request) {
                c.displayHeading("Operation Result");
                if (!resultAsTxt) {
                    c.displaySubResult("Output", c.dataToFileLink(response, "Output File"), false);
                } else {
                    c.displaySubResult("Output", response);
                }
            })
            .fail(c.displayDefaultFailure);
    };

    /* Process All-In-One (Adhoc Data) */
    var settings = {
        type:   "POST",
        url:    "/rest/serverengine/workflow/print/" + dmConfigId + "/" +
                    templateId + "/" + jcConfigId + "/" + ocConfigId +
                    "?async=" + async + "&createOnly=" + createOnly +
```

```
                        "&resultAsTxt=" + resultAsTxt + "&filename=" + file.name,
                data:           file,
                processData:    false,
                contentType:    "application/octet-stream"
            };
            if (!resultAsTxt && !async) settings.dataType = "file";
            if (printRange.length > 0) settings.url += "&printRange=" + printRange;
            $.ajax(settings)
                .done(function (response, status, request) {

                    if (async) {

                        var progress = null;
                        operationId = request.getResponseHeader("operationId");

                        $submitButton.prop("disabled", true);
                        $cancelButton.prop("disabled", false);

                        c.displayStatus("All-In-One Operation Successfully Submitted");
                        c.displayResult("Operation ID", operationId);

                        var getProgress = function () {
                            if (operationId !== null) {

                                /* Get Progress of Operation */
                                $.ajax({
                                    type:   "GET",
                                    cache:  false,
                                    url:    "/rest/serverengine/workflow/print/getProgress/" + operationId
                                })
                                    .done(function (response, status, request) {

                                        if (response !== "done") {
                                            if (response !== progress) {
                                                progress = response;
                                                $progressBar.attr("value", progress);
                                            }
                                            setTimeout(getProgress, 1000);
                                        } else {
                                            $progressBar.attr("value", (progress = 100));
                                            c.displayInfo("Operation Completed");
                                            getFinalResult();
                                            operationId = null;
                                            setTimeout(function () {
                                                $progressBar.attr("value", 0);
                                                $submitButton.prop("disabled", false);
                                                $cancelButton.prop("disabled", true);
                                            }, 100);
                                        }
                                    })
                                    .fail(c.displayDefaultFailure);
                            }
                        };
                        getProgress();
                    } else {
                        c.displayInfo("Operation Completed");
                        c.displayHeading("Operation Result");
                        if (!resultAsTxt) {
                            c.displaySubResult("Output", c.dataToFileLink(response, "Output File"), false);
                        } else {
                            c.displaySubResult("Output", response);
                        }
                    }
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output



Usage

To run the example, you first need to use the **Browse** button to select an appropriate **Data File** to use as an input using the selection dialog box.

Next, the following file based input fields can be referenced by **Managed File ID or Name**:

- Data Mapping configuration
- Template
- Output Creation preset

If required, a **Job Creation** preset can also be referenced by specifying a **Managed File ID or Name**.

The following options can be specified :

- **Asynchronous** – Run the All-In-One operation asynchronously with the operation progress being displayed and the option to cancel the running operation.
- **Create Only** – Create the output in server but do not send spool file to its final destination. In this example this would mean that the output files(s) would not be sent to the output directory specified in the output creation preset.
- **Get Result as Text** – Return the result as text specifically. In this example this would return the absolute path to the output file(s).
- **Print Range** – Restrict the print output to a specific range of records in the input data, not a specific range of pages.

Lastly, select the **Submit** button to start the All-In-One operation.

If running the operation *asynchronously*, once the operation has started processing, the Operation ID will be displayed in the **Results** area and the **Cancel** button will become enabled, giving you the option to cancel the running operation.

The progress of the operation will be displayed in the progress bar, and once the All-in-One operation has completed, the result will be returned and displayed to the **Results** area.

If running the operation *synchronously*, the result of the operation will simply be returned and displayed to the **Results** area.

The result returned will be the either the absolute path(s) of the output files or the output file itself.

> **Note:** If the result returned is expected to be file data, then then a download link will be displayed.

Further Reading

See the All-In-One Service page of the REST API Reference section for further detail.

REST API Reference

The PReS Connect REST API defines a number of RESTful services that facilitate various functionality within the server during workflow processing.

The following table is a summary of the services available in the PReS Connect REST API:

Service Name	Internal Name	Description
Authentication Service	*AuthenticationRestService*	This service exposes methods concerned with server security and authentication with the PReS Connect REST API. It includes methods to facilitate the following functions: • Authentication with the server (username & password / token based authorization)
Content Creation Service	*ContentCreationRestService*	This service exposes methods specific to the management of the content creation process for the *Print* context within the workflow. It includes methods to facilitate the following functions: • Creation, monitoring and cancellation of content creation operations for print using either a data set, data records or JSON as input • Getting a list of all the active operations on the server • Getting the result of a content creation operation for print • Creation of single record preview PDFs using either a data file, data record or JSON as input
Content Item Entity Service	*ContentItemEntityRestService*	This service exposes methods specific to the access and management of content item entities internal to the server. It includes methods to facilitate the following functions: • Getting the data record ID associated with a content item • Getting and updating of content item properties
Content Set Entity Service	*ContentSetEntityRestService*	This service exposes methods specific to the access and management of content set entities internal to the server. It includes methods to facilitate the following functions: • Getting all the content set IDs within the server • Getting the content item IDs contained within a content set • Getting the page details for a content set • Getting and updating of content set properties • Deletion of content sets from the server
Conversion Service	*ConversionRestService*	This service exposes methods specific to file conversions.

Service Name	Internal Name	Description
Data Record Entity Service	*DataRecordEntityRestService*	This service exposes methods specific to the access and management of data record entities internal to the server. It includes methods to facilitate the following functions: • Addition of new data records to a data set • Addition of new nested data records to a data record • Getting and updating of data record values • Getting and updating of data record properties
Data Set Entity Service	*DataSetEntityRestService*	This service exposes methods specific to the access and management of data set entities internal to the server. It includes methods to facilitate the following functions: • Getting all the data set IDs within the server • Getting the data record IDs contained within a data set • Getting and updating of data set properties • Deletion of data sets from the server
Data Mapping Service	*DataminingRestService*	This service exposes methods specific to the management of the data mapping process within the workflow. It includes methods to facilitate the following functions: • Creation, monitoring, cancellation and validation of data mapping operations using a data file as input • Creation, monitoring and cancellation of data mapping operations using a PDF/VT file as input • Getting a list of all the active operations on the server • Getting the result of a data mapping operation
Document Entity Service	*DocumentEntityRestService*	This service exposes methods specific to the access and management of document entities internal to the server. It includes methods to facilitate the following functions: • Getting and updating of document metadata properties
Document Set Entity Service	*DocumentSetEntityRestService*	This service exposes methods specific to the access and management of document set entities internal to the server. It includes methods to facilitate the following functions: • Getting the document IDs contained within a document set • Getting and updating of document set metadata properties
Content Creation (Email) Service	*EmailExportRestService*	This service exposes methods specific to the management of the content creation process for the *Email* context within the workflow. It includes methods to facilitate the following functions: • Creation, monitoring and cancellation of content creation operations for email using data records or JSON as input • Getting a list of all the active operations on the server • Getting the result of a content creation operation for email

Service Name	Internal Name	Description
Entity Service	*EntityRestService*	This service exposes methods specific to the querying and selection of data entities internal to the server. It includes methods to facilitate the following functions: • Finding of data entities in the server using specific search criteria
File Store Service	*FilestoreRestService*	This service exposes methods specific to the management of input and output files via the file store on the PReS Connect server. It includes methods to facilitate the following functions: • Uploading of data files and data mapping configurations to the file store • Uploading of templates to the File Store • Uploading of job creation and output creation presets to the file store • Download of managed files from the file store • Deletion of managed files from the file store
Content Creation (HTML) Service	*HTMLMergeRestService*	This service exposes methods specific to the management of the content creation process for the *Web* context within the workflow. It includes methods to facilitate the following functions: • Creation of HTML output for a specific data record or JSON input • Creation of HTML output from a template only (no input data) • Getting specific resources contained within a template
Job Creation Service	*JobCreationRestService*	This service exposes methods specific to the management of the job creation process within the workflow. It includes methods to facilitate the following functions: • Creation, monitoring and cancellation of job creation operations using either content sets or content items as input • Getting a list of all the active operations on the server • Getting the result of a job creation operation
Job Entity Service	*JobEntityRestService*	This service exposes methods specific to the access and management of job entities internal to the server. It includes methods to facilitate the following functions: • Getting the data record and content item IDs associated with a job • Getting the job segment IDs contained within a job • Getting and updating of job properties • Getting and updating of job metadata properties
Job Segment Entity Service	*JobSegmentEntityRestService*	This service exposes methods specific to the access and management of job segment entities internal to the server. It includes methods to facilitate the following functions: • Getting the document set IDs contained within a job segment • Getting and updating of job segment metadata properties

Service Name	Internal Name	Description
Job Set Entity Service	*JobSetEntityRestService*	This service exposes methods specific to the access and management of job set entities internal to the server. It includes methods to facilitate the following functions: • Getting all the job set IDs within the server • Getting the job IDs contained within a job set • Getting and updating of job set properties • Getting and updating of job set metadata properties • Deletion of job sets from the server
Output Creation Service	*OutputCreationRestService*	This service exposes methods specific to the management of the output creation process within the workflow. It includes methods to facilitate the following functions: • Creation, monitoring and cancellation of output creation operations using either a job set or jobs as input • Getting a list of all the active operations on the server • Getting the result of an output creation operation • Running of +PReS Enhance workflow configurations via the Weaver engine
All-In-One Service	*PrintRestService*	This service exposes methods specific to the management of the All-In-One process within the workflow. It includes methods to facilitate the following functions: • Creation, monitoring and cancellation of All-In-One operations using a variety of managed file and data entity input combinations • Creation of synchronous All-In-One operations using a data file as input • Getting a list of all the active operations on the server • Getting the result of an All-In-One operation

All-In-One Service

The following table is a summary of the resources and methods available in the All-In-One service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/workflow/print	GET
Process All-In-One (JSON)	/rest/serverengine/workflow/print/submit	POST
Process All-In-One (Adhoc Data)	/rest/serverengine/workflow/print/{dmConfigId}/{templateId}/{jcConfigId}/{ocConfigId}	POST
Get All Operations	/rest/serverengine/workflow/print/getOperations	GET
Get Progress of Operation	/rest/serverengine/workflow/print/getProgress/{operationId}	GET
Get Result of Operation	/rest/serverengine/workflow/print/getResult/{operationId}	POST
Get Result of Operation (as Text)	/rest/serverengine/workflow/print/getResultTxt/{operationId}	POST
Get Managed Result of Operation	/rest/serverengine/workflow/print/getManagedResult/{operationId}	POST
Cancel an Operation	/rest/serverengine/workflow/print/cancel/{operationId}	POST
Service Version	/rest/serverengine/workflow/print/version	GET

Cancel an Operation

Requests the cancellation of a running All-In-One operation of a specific operation ID.

Request takes no content, and on success returns a response with no content.

Request

	POST
	/rest/serverengine/workflow/print/cancel/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path parameters

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of All-In-One operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
204 *No Content*	–	–	–	Operation cancellation requested

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Get All Operations

Returns a list of all the workflow operations actively running on the Server.

Request takes no content, and on success returns a response containing a JSON Operations List of all the actively running operations.

Request

	GET
	/rest/serverengine/workflow/print/getOperations
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Operations List of all the actively running operations in Server	application/json	–	List of actively running operations successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Managed Result of Operation

Retrieves the Managed File ID of the final result of a completed Output Creation operation of a specific operation ID.

Request takes no content, and on success returns the Managed File ID of the output (file or directory) in the File Store.

Request

	POST
	/rest/serverengine/workflow/print/getManagedResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of All-In-One operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID of the output (file or directory) in the File Store	text/plain	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Get Progress of Operation

Retrieves the progress of a running All-In-One operation of a specific operation ID.

Request takes no content, and on success returns a response containing the current value of operation progress (values ranging from 0 – 100, followed by the value of 'done' on completion).

Request

	GET
	/rest/serverengine/workflow/print/getProgress/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of All-In-One operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Progress value of All-In-One operation	text/plain	–	Progress of operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Result of Operation

Retrieves the final result of a completed All-In-One operation of a specific operation ID.

Request takes no content, and on success returns a response (depending on the All-In-One configuration) containing either:

- the ID of the Data Set, Content Set or Job Set entity produced, or
- the absolute paths of the final output files produced (multiple spool files) or the content of a final output file (single spool file).

Request

	POST
	/rest/serverengine/workflow/print/getResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of All-In-One operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Either: • ID of the Data Set, Content Set or Job Set, or • Absolute Paths of the Output Files or the Output File itself	application/octet-stream	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Get Result of Operation (as Text)

Retrieves the final result of a completed All-In-One operation of a specific operation ID.

Request takes no content, and on success returns a response (depending on the All-In-One configuration) containing either:

- the ID of the Data Set, Content Set or Job Set entity produced, or
- the absolute path or paths of the final output file or files produced (single or multiple spool files respectively).

Request

	POST
	/rest/serverengine/workflow/print/getResultTxt/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of All-In-One operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Either: • ID of the Data Set, Content Set or Job Set • Absolute Path(s) of the Output File(s)	text/plain	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Process All-In-One (JSON)

Submits a request to initiate a new All-In-One operation.

Request takes a JSON All-In-One Configuration as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/print/submit
	JSON All-In-One Configuration containing workflow processes/parameters
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new All-In-One operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Required Input resource/file not found in File Store
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	General error with running the All-In-One Process or a specific error relating to an individual workflow process (see error description)

Process All-In-One (Adhoc Data)

Submits a request to initiate a new All-In-One operation using pre-existing inputs, with the exception of input data, which is submitting along with the request.

Request takes binary file data as content, and on success will return one of two responses depending on the type of operation:

- **Asynchronous** – response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation (Default)
- **Synchronous** – response (depending on the parameters specified) containing either:
  - the absolute paths of the final output files produced (multiple spool files) or the content of a final output file (single spool file)
  - the absolute path or paths of the final output file or files produced (single or multiple spool files respectively) (Get Result as Text Only)

Request

	POST
	/rest/serverengine/workflow/print/{dmConfigId}/{templateId}/{jcConfigId}/{ocConfigId}
	Data File (File)
	application/octet-stream
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dmConfigId	–	–	–	The Managed File ID (or Name) of the Data Mapping configuration in File Store
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store
jcConfigId	Optional – the value of "0" can be specified if no preset is to be used	–	–	The Managed File ID (or Name) of the Job Creation Preset in File Store

Name	Required	Type	Default Value	Description
ocConfigId	–	–	–	The Managed File ID (or Name) of the Output Creation Preset in File Store

Query

Name	Required	Type	Default Value	Description
async	–	–	true	Whether to run the operation asynchronously
resultAsTxt	–	–	false	Whether to retrieve the result as text (Synchronous Only)
createOnly	–	–	false	Whether output is to be only created in the server and not sent to it's final destination
printRange	–	–	–	A specific range of records in the input data file to restrict the print output to
filename	–	–	–	The file name of the data file to be uploaded

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new All-In-One operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Data file successfully uploaded to File Store and creation of new operation successful

Success (Synchronous)

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Absolute Paths of the Output Files or the Output File itself	application/octet-stream	–	Data file uploaded to File Store and a new operation was created and completed successfully with the result returned

Success (Synchronous + Get Result as Text)

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Absolute Path(s) of the Output File(s)	text/plain	–	Data file uploaded to File Store and a new operation was created and completed successfully with the result returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Unable to locate one or more inputs in File Store with Managed File ID(s) and/or Name(s) specified
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	General error with running the All-In-One Process or a specific error relating to the uploading of the data file or an individual workflow process (see error description)

Service Handshake

Queries the availability of the All-In-One service.

Request

	GET
	/rest/serverengine/workflow/print
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: PrintRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the All-In-One service.

Request

	GET
	/rest/serverengine/workflow/print/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Authentication Service

The following table is a summary of the resources and methods available in the Authentication service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/authentication	GET
Authenticate/Login to Server	/rest/serverengine/authentication/login	POST
Service Version	/rest/serverengine/authentication/version	GET

Authenticate/Login to Server

Submits an authentication request (using credentials) to the Connect server and if successful provides access to the various other REST API services available.

Request takes no content, but requires an additional **Authorization** header which contains a base64 encoded set of credentials (username and password). On success, the response will return an authorization token which can then be used as an additional **auth_token** header in any future requests made to the REST API services.

If server security settings are enabled and a request is made to any resource of any service in the REST API, if that request contains neither an **auth_token** header or an **Authorization** header, then the response will come back as Unauthorized and will contain an additional **WWW-Authenticate** response header.

Request

	POST
	/rest/serverengine/authentication/login
	–
	–
	Authorization – Basic Authentication (Username and Password) credentials (Base64 encoded)

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Authorization Token	text/plain	**Token-Expires-In** - The number of seconds the token is valid	Server authentication successful, new authorization token generated.

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication has failed. Response when no basic authentication credentials have been specified in the request headers.
401 *Unauthorized*	–	–	–	Server authentication has failed. Response when the basic authentication credentials specified in the request headers are invalid.

Service Handshake

Queries the availability of the Authentication service.

Request

	GET
	/rest/serverengine/authentication
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: AuthenticationRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Authentication service.

Request

	GET
	/rest/serverengine/authentication/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Content Creation (Email) Service

The following table is a summary of the resources and methods available in the Content Creation (Email) service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/workflow/contentcreation/email	GET
Process Content Creation (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/email/{templateId}	POST
Process Content Creation (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/email/{templateId}	POST
Process Content Creation (By Data Set ID) (JSON)	/rest/serverengine/workflow/contentcreation/email/{templateId}/{dataSetId}	POST
Get All Operations	/rest/serverengine/workflow/contentcreation/email/getOperations	GET
Get Progress of Operation	/rest/serverengine/workflow/contentcreation/email/getProgress/{operationId}	GET
Get Result of Operation	/rest/serverengine/workflow/contentcreation/email/getResult/{operationId}	POST
Cancel an Operation	/rest/serverengine/workflow/contentcreation/email/cancel/{operationId}	POST
Service Version	/rest/serverengine/workflow/contentcreation/email/version	GET

Cancel an Operation

Requests the cancellation of a running Content Creation (Email) operation of a specific operation ID.

Request takes no content, and on success returns a response with no content.

Request

	POST
	/rest/serverengine/workflow/contentcreation/email/cancel/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path parameters

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Content Creation (Email) operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
204 *No Content*	–	–	–	Operation cancellation requested

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Get All Operations

Returns a list of all the workflow operations actively running on the Server.

Request takes no content, and on success returns a response containing a JSON Operations List of all the actively running operations.

Request

	GET
	/rest/serverengine/workflow/contentcreation/email/getOperations
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Operations List of all the actively running operations in Server	application/json	–	List of actively running operations successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Progress of Operation

Retrieves the progress of a running Content Creation (Email) operation of a specific operation ID.

Request takes no content, and on success returns a response containing the current value of operation progress (values ranging from 0 – 100, followed by the value of 'done' on completion).

Request

	GET
	/rest/serverengine/workflow/contentcreation/email/getProgress/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Content Creation (Email) operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Progress value of Content Creation (Email) operation	text/plain	–	Progress of operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Result of Operation

Retrieves the final result of a completed Content Creation (Email) operation of a specific operation ID.

Request takes no content, and on success returns a response containing a JSON Email Output List of all the email output messages created in the File Store.

Alternatively, if the operation was to create email output directly to a SMTP mail server, then a response containing a report on the number of emails that were successfully sent will be returned instead.

Request

	POST
	/rest/serverengine/workflow/contentcreation/email/getResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Content Creation (Email) operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	"JSON Email Output List" on page 53 of the email message output in File Store	text/plain	–	Result of completed operation successfully retrieved Response when the email parameters specified in the *original* request that created the operation **does not** contain a **host** value specifying the network address or name of a SMTP mail server

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of Content Creation (Email) Operation (with successful email count) (e.g. *"3 of 3 emails sent"*)	text/plain	–	Result of completed operation successfully retrieved Response when the email parameters specified in the *original* request that created the operation contained a **host** value specifying the network address or name of a SMTP mail server

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Context not found / Section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Process Content Creation (By Data) (JSON)

Submits a request to initiate a new Content Creation (Email) operation.

Request takes a JSON Record Data List (with Email Parameters) of the data values for one or more Data Records as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/contentcreation/email/{templateId}
	JSON Record Data List (with Email Parameters) specifying a list of data values for the Data Record(s) and parameters to be used for content creation
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store

Query

Name	Required	Type	Default Value	Description
section	–	–	Default section in template	The Section of the Email context to export

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Content Creation (Email) operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store

Process Content Creation (By Data Record) (JSON)

Submits a request to initiate a new Content Creation (Email) operation.

Request takes a JSON Identifier List (with Email Parameters) of Data Record IDs as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/contentcreation/email/{templateId}
	JSON Identifier List (with Email Parameters) specifying a list of Data Record entity IDs and parameters to be used for content creation
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store

Query

Name	Required	Type	Default Value	Description
section	–	–	Default section in template	The Section of the Email context to export

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Content Creation (Email) operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template or Data Record entity not found in File Store/Server

Service Handshake

Queries the availability of the Content Creation (Email) service.

Request

	GET
	/rest/serverengine/workflow/contentcreation/email
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: EmailExportRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Content Creation (Email) service.

Request

	GET
	/rest/serverengine/workflow/contentcreation/email/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Content Creation (HTML) Service

The following table is a summary of the resources and methods available in the Content Creation (HTML) service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/workflow/contentcreation/html	GET
Process Content Creation (By Data Record)	/rest/serverengine/workflow/contentcreation/html/{templateId}/{dataRecordId: [0-9]+}	GET
Process Content Creation (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/html/{templateId}/{dataRecordId: [0-9]+}	POST
Process Content Creation (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/html/{templateId}	POST
Process Content Creation (No Data)	/rest/serverengine/workflow/contentcreation/html/{templateId}	POST
Get Template Resource	/rest/serverengine/workflow/contentcreation/html/{templateId}/{relPath: .+}	GET
Service Version	/rest/serverengine/workflow/contentcreation/html/version	GET

Get Template Resource

Submits a request to retrieve a resource from a template stored in the File Store.

Request takes no content, and on success returns a response containing the resource from the template.

Request

	GET
	/rest/serverengine/workflow/contentcreation/html/{templateId}/{relPath: .+}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store
relPath	–	–	–	The relative path to the resource within the template

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Resource located at the relative path within template	(Depends on Resource requested)	–	Resource successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Unable to open resource within template or resource doesn't exist
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store
500 *Internal Server Error*	–	–	–	Unable to open template or template doesn't exist

Process Content Creation (By Data) (JSON)

Submits a request to create new HTML content for the Web context.

Request takes a JSON Record Data List of the data values for the Data Record as content, and on success returns a response containing the HTML output produced, specific to the record data and parameters specified.

Request

	POST
	/rest/serverengine/workflow/contentcreation/html/{templateId}
	JSON Record Data List specifying a list of data values for the Data Record
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store

Query

Name	Required	Type	Default Value	Description
section	–	–	Default section in template	The section within the Web context to create
inline	–	–	NONE	The inline mode to be used in the creation of content. Possible values: ◦ `NONE` - no inlining ◦ `CSS` - converts style rules to inline styles on elements ◦ `ALL` - inline all resources ◦ `LOCAL` - inline local resources; remote resources remain external
cssSelector	No	String		A CSS selector for the creation of only a specific HTML element within the template

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	The entire HTML output for the Data Record, or the inner HTML of the first element that matches the given CSS selector, or an empty string if the CSS selector produced no results	text/html	–	Output created successfully

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Web context not found / Web section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store/Server
500 *Internal Server Error*	–	–	–	Content Creation Error: Web context in template not found / Invalid CSS selector.

Process Content Creation (By Data Record)

Submits a request to create new HTML content for the Web context.

Request takes no content, and on success returns a response containing the HTML output produced, specific to the Data Record and parameters specified.

Request

	GET
	/rest/serverengine/workflow/contentcreation/html/{templateId}/{dataRecordId: [0-9]+}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store
dataRecordId	–	–	–	The ID of the Data Record entity in Server

Query

Name	Required	Type	Default Value	Description
section	–	–	Default section in template	The section within the Web context to create
inline	–	–	NONE	The inline mode to be used in the creation of content. Possible values: ◦ `NONE` - no inlining ◦ `CSS` - converts style rules to inline styles on elements ◦ `ALL` - inline all resources ◦ `LOCAL` - inline local resources; remote resources remain external
cssSelector	No	String		A CSS selector for the creation of only a specific HTML element within the template

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	The entire HTML output for the Data Record, or the inner HTML of the first element that matches the given CSS selector, or an empty string if the CSS selector produced no results	text/html	–	Output created successfully

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Web context not found / Web section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template or Data Record entity not found in File Store/Server
500 *Internal Server Error*	–	–	–	Content Creation Error: Data Record not found / Web context in template not found / Invalid CSS selector.

Process Content Creation (By Data Record) (JSON)

Submits a request to create new HTML content for the Web context.

Request takes a JSON HTML Parameters as content, and on success returns a response containing the HTML output produced, specific to the Data Record and parameters specified.

Request

	POST
	/rest/serverengine/workflow/contentcreation/html/{templateId}/{dataRecordId: [0-9]+}
	JSON HTML Parameters specifying parameters to be used for content creation
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store
dataRecordId	–	–	–	The ID of the Data Record entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	The entire HTML output for the Data Record, or the inner HTML of the first element that matches the given CSS selector, or an empty string if the CSS selector produced no results	text/html	–	Output created successfully

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Web context not found / Web section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template or Data Record entity not found in File Store/Server
500 *Internal Server Error*	–	–	–	Content Creation Error: Data Record not found / Web context in template not found / Invalid CSS selector

Process Content Creation (No Data)

Submits a request to create new HTML content for the Web context.

Request takes no content, and on success returns a response containing the HTML output produced, using only the template, no input data and specific to the parameters specified.

Request

	POST
	/rest/serverengine/workflow/contentcreation/html/{templateId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store

Query

Name	Required	Type	Default Value	Description
section	–	–	Default section in template	The section within the Web context to create
inline	–	–	NONE	The inline mode to be used in the creation of content. Possible values: ◦ `NONE` - no inlining ◦ `CSS` - converts style rules to inline styles on elements ◦ `ALL` - inline all resources ◦ `LOCAL` - inline local resources; remote resources remain external
cssSelector	No	–	–	A CSS selector for the creation of only a specific HTML element within the template

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	The entire HTML output for the Data Record, or the inner HTML of the first element that matches the given CSS selector, or an empty string if the CSS selector produced no results	text/html	–	Output created successfully

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Web context not found / Web section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store
500 *Internal Server Error*	–	–	–	Content Creation Error: Web context in template not found / Invalid CSS selector.

Service Handshake

Queries the availability of the Content Creation (HTML) service.

Request

	GET
	/rest/serverengine/workflow/contentcreation/html
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: HTMLMergeRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Content Creation (HTML) service.

Request

	GET
	/rest/serverengine/workflow/contentcreation/html/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Content Creation Service

The following table is a summary of the resources and methods available in the Content Creation service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/workflow/contentcreation	GET
"Process Content Creation (By Data Set)" on page 388	/rest/serverengine/workflow/contentcreation/{templateId}/{dataSetId}	POST
Process Content Creation (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/{templateId}	POST
Process Content Creation (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/{templateId}	POST
Create Preview PDF (By Data Record)	/rest/serverengine/workflow/contentcreation/pdfpreviewdirect	GET
"Create Preview PDF (By Data)" on page 396	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}/{dmConfigId}	POST
Create Preview PDF (By Data) (JSON)	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}	POST
Create Preview Image (By Data Record) (JSON)	/rest/serverengine/workflow/contentcreation/imagepreview	POST
"Create Preview Image (By Data) (JSON)" on page 401	/rest/serverengine/workflow/contentcreation/imagepreview/{templateId}	POST
Get All Operations	/rest/serverengine/workflow/contentcreation/getOperations	GET
Get Progress of Operation	/rest/serverengine/workflow/contentcreation/getProgress/{operationId}	GET
Get Result of Operation	/rest/serverengine/workflow/contentcreation/getResult/{operationId}	POST
Get Managed Result of Operation	/rest/serverengine/workflow/outputcreation/getManagedResult/{operationId}	POST
Cancel an Operation	/rest/serverengine/workflow/contentcreation/cancel/{operationId}	POST
Service Version	/rest/serverengine/workflow/contentcreation/version	GET

Service Handshake

Queries the availability of the Content Creation service.

Request

	GET
	/rest/serverengine/workflow/contentcreation
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: ContentCreationRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Process Content Creation (By Data Set)

Submits a request to initiate a new Content Creation operation.

Request takes a JSON Parameters object, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/contentcreation/{templateId}/{dataSetId}
	A "JSON Parameters" on page 90 object describing a list of runtime parameters.
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store
dataSetId	–	–	–	The ID of the Data Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Content Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template or Data Set entity not found in File Store/Server

Process Content Creation (By Data Record) (JSON)

Submits a request to initiate a new Content Creation operation.

Request takes a JSON Identifier List (with Runtime Parameters) of Data Record IDs as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/contentcreation/{templateId}
	JSON Identifier List (with Runtime Parameters) specifying a list of Data Record entity IDs
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Content Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store

Process Content Creation (By Data) (JSON)

Submits a request to initiate a new Content Creation operation.

Request takes a JSON Record Data List of the data values for one or more Data Records as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/contentcreation/{templateId}
	JSON Record Data List specifying a list of data values for the Data Record(s)
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Content Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store

Create Preview PDF (By Data Record)

Submits a request to create a preview PDF of the print output for a single data record.

Request takes no content, and on success returns a response containing the Managed File ID for the newly created preview PDF file.

Request

	GET
	/rest/serverengine/workflow/contentcreation/pdfpreviewdirect
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Query

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store
dataRecordId	–	–	–	The ID of the Data Record entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID	text/plain	–	Creation of preview PDF in File Store successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Unable to open resource within template or resource doesn't exist / Context not found / Section not found

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template or Data Record entity not found in File Store/Server

Create Preview PDF (By Data)

Submits a request to create a preview PDF of the print output for a single data record.

Request takes binary file data as content, and on success returns a response containing the Managed File ID for the newly created preview PDF file.

Request

	POST
	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}/{dmConfigId}
	Data File (File)
	application/octet-stream
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store
dmConfigId	–	–	–	The Managed File ID (or Name) of the Data Mapping configuration in File Store

Query

Name	Required	Type	Default Value	Description
persist	–	–	true	Whether the Data Record produced will be persisted in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID	text/plain	–	Creation of preview PDF in File Store successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Unable to open resource within template or resource doesn't exist / Context not found / Section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template, Data Mapping configuration or Data Record entity not found in File Store/Server

Create Preview PDF (By Data) (JSON)

Submits a request to create a preview PDF of the print output for a single data record.

Request takes a JSON Record Data List of the data values for the Data Record as content, and on success returns a response containing the Managed File ID for the newly created preview PDF file.

Request

	POST
	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}
	JSON Record Data List specifying a list of data values for the Data Record
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
templateId	–	–	–	The Managed File ID (or Name) of the template in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID	text/plain	–	Creation of preview PDF in File Store successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Unable to open resource within template or resource doesn't exist / Context not found / Section not found

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store

Create Preview Image (By Data Record) (JSON)

Submits a request to create a preview image of the print, email, or web

Request takes a set of JSON Image Parameters as content, and on success returns a response containing the image output produced, specific to the record data and parameters specified.

Request

	POST
	/rest/serverengine/workflow/contentcreation/imagepreview
	JSON Image Parameters specifying parameters to be used for content creation
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Query Parameters

The following lists the query parameters accepted by this method:

Name	Required	Type	Default Value	Description
templateId	Yes	String	–	The Managed File ID (or Name) of the template in File Store
dataRecordId	Yes	Number	–	The ID of the Data Record entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Preview Image	image/jpeg	–	Creation of a preview image successful. Response when the image parameters specified contain: • a **type** value of *JPG* or *JPEG* • an **archive** value and **pages** value that would result in a single page
200 *OK*	Preview Image	image/png	–	Creation of a preview image successful. Response when the image parameters specified contain: • a **type** value of *PNG* • an **archive** value and **pages** value that would result in a single page
200 *OK*	Preview Images (Zipped)	application/zip	–	Creation of preview images successful. Response when the image parameters specified contain: • an **archive** value and **pages** value that would result in a multiple pages

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Context not found / Section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	JSON Error specifying error message	application/zip	–	Template or Data Record entity not found in File Store/Server

Create Preview Image (By Data) (JSON)

Submits a request to create a preview image of the print, email, or web output for a single data record.

Request takes a JSON Record Data List (with Image Parameters) of the data values for the Data Record as content, and on success returns a response containing the image output produced, specific to the record data and parameters specified.

Request

	POST
	/rest/serverengine/workflow/contentcreation/imagepreview/{templateId}
	JSON Record Data List (with Image Parameters) of the data values for the Data Record
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path Parameters

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
templateId	Yes	String	–	The Managed File ID (or Name) of the template in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Preview Image	image/jpeg	–	Creation of a preview image successful. Response when the image parameters specified contain: • a **type** value of *JPG* or *JPEG* • an **archive** value and **pages** value that would result in a single page

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Preview Image	image/png	–	Creation of a preview image successful. Response when the image parameters specified contain: • a **type** value of *PNG* • an **archive** value and **pages** value that would result in a single page
200 *OK*	Preview Images (Zipped)	application/zip	–	Creation of preview images successful. Response when the image parameters specified contain: • an **archive** value and **pages** value that would result in a multiple pages

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Context not found / Section not found
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	JSON Error specifying error message	application/zip	–	Template found in File Store

Get All Operations

Returns a list of all the workflow operations actively running on the Server.

Request takes no content, and on success returns a response containing a JSON Operations List of all the actively running operations.

Request

	GET
	/rest/serverengine/workflow/contentcreation/getOperations
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Operations List of all the actively running operations in Server	application/json	–	List of actively running operations successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Progress of Operation

Retrieves the progress of a running Content Creation operation of a specific operation ID.

Request takes no content, and on success returns a response containing the current value of operation progress (values ranging from 0 – 100, followed by the value of 'done' on completion).

Request

	GET
	/rest/serverengine/workflow/contentcreation/getProgress/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Content Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Progress value of Content Creation operation	text/plain	–	Progress of operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Result of Operation

Retrieves the final result of a completed Content Creation operation of a specific operation ID.

Request takes no content, and on success returns a response containing the IDs of the Content Sets produced.

Request

	POST
	/rest/serverengine/workflow/contentcreation/getResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Content Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Content Set IDs	text/plain	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Context not found / Section not found

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Get Managed Result of Operation

Retrieves the Managed File ID of the final result of a completed Output Creation operation of a specific operation ID.

Request takes no content, and on success returns the Managed File ID of the output (file or directory) in the File Store.

Request

	POST
	/rest/serverengine/workflow/outputcreation/getManagedResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Content Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID of the output (file or directory) in the File Store	text/plain	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Cancel an Operation

Requests the cancellation of a running Content Creation operation of a specific operation ID.

Request takes no content, and on success returns a response with no content.

Request

	POST
	/rest/serverengine/workflow/contentcreation/cancel/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path parameters

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Content Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
204 *No Content*	–	–	–	Operation cancellation requested

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Service Version

Returns the version of the Content Creation service.

Request

	GET
	/rest/serverengine/workflow/contentcreation/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Content Item Entity Service

The following table is a summary of the resources and methods available in the Content Item Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/entity/contentitems	GET
Get Data Record for Content Item	/rest/serverengine/entity/contentitems/{contentItemId}/datarecord	GET
Get Content Item Properties	/rest/serverengine/entity/contentitems/{contentItemId}/properties	GET
Update Content Item Properties	/rest/serverengine/entity/contentitems/{contentItemId}/properties	PUT
Update Multiple Content Item Properties	/rest/serverengine/entity/contentitems/properties	PUT
Service Version	/rest/serverengine/entity/contentitems/version	GET

Get Content Item Properties

Returns a list of the properties for a specific Content Item entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the properties for the Content Item.

Request

	GET
	/rest/serverengine/entity/contentitems/{contentItemId}/properties
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentItemId	–	–	–	ID of the Content Item entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of properties for Content Item	application/json	–	Content Item entity properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Data Record for Content Item

Returns the ID of the corresponding Data Record for a specific Content Item entity.

Request takes no content, and on success returns a response containing a JSON Data Record Identifier for the Data Record of the Content Item.

Request

	GET
	/rest/serverengine/entity/contentitems/{contentItemId}/datarecord
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentItemId	–	–	–	ID of the Content Item entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Data Record Identifier for the Data Record of Content Item	application/json	–	Data Record Identifier returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Handshake

Queries the availability of the Content Item Entity service.

Request

	GET
	/rest/serverengine/entity/contentitems
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: ContentItemEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Content Item Entity service.

Request

	GET
	/rest/serverengine/entity/contentitems/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Content Item Properties

Submits a request to update (and replace) the properties for a specific Content Item entity in the Server.

Request takes a JSON Name/Value List as content (the Content Item ID and the new properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/contentitems/{contentItemId}/properties
	JSON Name/Value List of properties for Content Item
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentItemId	–	–	–	ID of the Content Item entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Content Item	text/plain	–	Update of Content Item properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Content Item ID mismatch in JSON

Update Multiple Content Item Properties

Submits a request to update one or more properties for one or more Content Item entities in the Server.

Request takes JSON Name/Value Lists as content (each with the Content Item ID and the new properties), and on success returns a response containing no content.

Request

	PUT
	/rest/serverengine/entity/contentitems/properties
	JSON Name/Value Lists of the properties of the Content Items
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	–	–	–	Properties of Content Item entities successfully updated

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Content Set Entity Service

The following table is a summary of the resources and methods available in the Content Set Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
"Service Handshake" below	/rest/serverengine/entity/contentsets	GET
Get All Content Sets	/rest/serverengine/entity/contentsets	GET
Get Content Items for Content Set	/rest/serverengine/entity/contentsets/{contentSetId}	GET
Get Page Details for Content Set	/rest/serverengine/entity/contentsets/{contentSetId}/pages	GET
Delete Content Set Entity	/rest/serverengine/entity/contentsets/{contentSetId}/delete	POST
Get Content Set Properties	/rest/serverengine/entity/contentsets/{contentSetId}/properties	GET
Update Content Set Properties	/rest/serverengine/entity/contentsets/{contentSetId}/properties	PUT
Service Version	/rest/serverengine/entity/contentsets/version	GET

Service Handshake

Queries the availability of the Content Set Entity service.

Request

	GET
	/rest/serverengine/entity/contentsets
	–
	–
	text/plain
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: ContentSetEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Delete Content Set Entity

Submits a request for a specific Content Set entity to be marked for deletion from the Server.

Request takes no content, and on success returns a response containing the result of the request for deletion (*"true"* or *"false"*).

Request

	POST
	/rest/serverengine/entity/contentsets/{contentSetId}/delete
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentSetId	–	–	–	ID of the Content Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request for Content Set removal	text/plain	–	Deletion of Content Set successfully requested from Server (response of "true" for success or "false" for failure)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get All Content Sets

Returns a list of all the Content Set entities currently contained within the Server.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Content Sets.

Request

	GET
	/rest/serverengine/entity/contentsets
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Content Sets in Server	application/json	–	Identifier List of Content Sets returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Content Items for Content Set

Returns a list of all the Content Item entities (and their corresponding Data Record entities) contained within a specific Content Set entity.

Request takes no content, and on success returns a response containing a JSON Content Item Identifier List of all the Content Items in the Content Set.

Request

	GET
	/rest/serverengine/entity/contentsets/{contentSetId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentSetId	–	–	–	ID of the Content Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Content Item Identifier List of all the Content Items in Content Set	application/json	–	Content Item Identifier List returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Content Set Properties

Returns a list of the properties for a specific Content Set entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the properties for the Content Set.

Request

	GET
	/rest/serverengine/entity/contentsets/{contentSetId}/properties
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentSetId	–	–	–	ID of the Content Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of properties for Content Set	application/json	–	Content Set entity properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Page Details for Content Set

Returns the page details for a specific Content Set entity, as either a summary or a list (broken down by Content Item entity).

Request takes no content, and on success returns a response containing either:

- JSON Page Details Summary of the page details for the Content Set
- JSON Page Details List of the page details for each Content Item in the Content Set

Request

	GET
	/rest/serverengine/entity/contentsets/{contentSetId}/pages
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentSetId	–	–	–	ID of the Content Set entity in Server

Query

Name	Required	Type	Default Value	Description
detail	–	–	False	Return a list of details for each Content Item in the Content Set instead of a summary

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Page Details Summary containing page details for Content Set	application/json	–	Content Set entity page details successfully retrieved
200 *OK*	JSON Page Details List containing page details for each Content Item in Content Set	application/json	–	Content Set entity page details successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Content Set Entity service.

Request

	GET
	/rest/serverengine/entity/contentsets/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Content Set Properties

Submits a request to update (and replace) the properties for a specific Content Set entity in the Server.

Request takes a JSON Name/Value List as content (the Content Set ID and the new properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/contentsets/{contentSetId}/properties
	JSON Name/Value List of properties for Content Set
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
contentSetId	–	–	–	ID of the Content Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Content Set	text/plain	–	Update of Content Set properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Content Set ID mismatch in JSON

Conversion Service

The following table is a summary of the resources and methods available in the Conversion service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Rasterize File	/rest/serverengine/workflow/rasterize/{dataFileID}	POST

Rasterize File

Converts a PDF, PS, AFP or PCL file into a bitmap. The conversion can be done for a single page or for multiple pages.

Request takes JSON Bitmap Parameters JSON Bitmap Parameters as content, and on success returns the converted page as a PNG or JPEG file, or returns a ZIP file with the converted images (PNG/JPEG) if rasterization is requested for more than one page.

Request

	POST
	/rest/serverengine /workflow/rasterize/{dataFileID}
	JSON Bitmap Parameters
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataFileId	Yes	Number	–	The Managed File ID (or Name) of a PDF, PS, AFP or PCL file in the File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	A stream with a JPEG/PNG image or a ZIP file with JPEG/PNG images	application/zip, image/jpeg, or image/png		Bitmap creation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json		ID of uploaded PDF, PS, PCL, or AFP data file cannot be found in filestore

Data Mapping Service

The following table is a summary of the resources and methods available in the Data Mapping service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/workflow/datamining	GET
Create Content Set	/rest/serverengine/workflow/datamining/createcontentset/{dataFileID}	POST
Process Data Mapping	/rest/serverengine/workflow/datamining/{configId}/{dataFileId}	POST
Process Data Mapping (JSON)	/rest/serverengine/workflow/datamining/{configId}	POST
Process Data Mapping (PDF/VT to Data Set)	/rest/serverengine/workflow/datamining/pdfvtds/{dataFileId}	POST
Process Data Mapping (PDF/VT to Content Set)	/rest/serverengine/workflow/datamining/pdfvtcs/{dataFileId}	POST
Get All Operations	/rest/serverengine/workflow/datamining/getOperations	GET
Get Progress of Operation	/rest/serverengine/workflow/datamining/getProgress/{operationId}	GET
Get Result of Operation	/rest/serverengine/workflow/datamining/getResult/{operationId}	POST
Cancel an Operation	/rest/serverengine/workflow/datamining/cancel/{operationId}	POST
Service Version	/rest/serverengine/workflow/datamining/version	GET

Cancel an Operation

Requests the cancellation of a running Data Mapping operation of a specific operation ID.

Request takes no content, and on success returns a response with no content.

Request

	POST
	/rest/serverengine/workflow/datamining/cancel/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path parameters

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Data Mapping operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
204 *No Content*	–	–	–	Operation cancellation requested

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Create Content Set

Submits a request to initiate a new Data Mapping operation.

The Data Mapping operation generates a Data Set, based on the provided data mapping configuration and PDF/PS/AFP/PCL data file, and a Content Set based on the Data Set.
The data file is consumed according to the data mapping configuration. The resulting data records are then used as background in Content Items. The Content Items are created without a template.

Request takes a "JSON Identifier (for Content Creation)" on page 61, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/datamining/createcontentset/{dataFileID}
	"JSON Identifier (for Content Creation)" on page 61
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataFileID	–	–	–	The ID of the PDF/PS/PCL/AFP data file in the File Store that will be consumed according to the data mapping configuration to create a Content Set.

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Data mapping configuration not made for PDF files.
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Data file or data mapping configuration not found in File Store

Get All Operations

Returns a list of all the workflow operations actively running on the Server.

Request takes no content, and on success returns a response containing a JSON Operations List of all the actively running operations.

Request

	GET
	/rest/serverengine/workflow/datamining/getOperations
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Operations List of all the actively running operations in Server	application/json	–	List of actively running operations successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Progress of Operation

Retrieves the progress of a running Data Mapping operation of a specific operation ID.

Request takes no content, and on success returns a response containing the current value of operation progress (values ranging from 0 – 100, followed by the value of 'done' on completion).

Request

	GET
	/rest/serverengine/workflow/datamining/getProgress/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Data Mapping operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Progress value of Data Mapping operation	text/plain	–	Progress of operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Result of Operation

Retrieves the final result of a completed Data Mapping operation of a specific operation ID.

Request takes no content, and on success returns a response containing the ID of the Data Set produced (or Content Set for a PDF/VT to Content Set specific data mapping operation).

Alternatively, if the operation was to only **validate** the data mapping, then a response containing a JSON Data Mapping Validation Result will be returned instead.

Request

	POST
	/rest/serverengine/workflow/datamining/getResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Data Mapping operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Data Set ID (or Content Set ID)	text/plain	–	Result of completed operation successfully retrieved

Success (validate)

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Data Mapping Validation Result	text/plain	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Process Data Mapping

Submits a request to initiate a new Data Mapping operation.

Request takes an optional JSON Data Mapping Parameter, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/datamining/{configId}/{dataFileId}
	(Optional) JSON Data Mapping Parameter
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
configId	–	–	–	The Managed File ID (or Name) of the Data Mapping configuration in File Store
dataFileId	–	–	–	The Managed File ID (or Name) of the PDF/VT data file in File Store

Query

Name	Required	Type	Default Value	Description
validate	–	–	false	Only validate the Data Mapping operation to check for mapping errors

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Data file or Data Mapping Configuration not found in File Store
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Error (Validate)

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Data file or Data Mapping Configuration not found in File Store

Process Data Mapping (JSON)

Submits a request to initiate a new Data Mapping operation.

Request takes a JSON Identifier (for Data Mapping) as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/datamining/{configId}
	JSON Identifier (for Data Mapping)
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
configId	–	–	–	Managed File ID (or Name) of the Data Mapping configuration in File Store

Query

Name	Required	Type	Default Value	Description
validate	–	–	false	Only validate the Data Mapping operation to check for mapping errors

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Data file or Data Mapping Configuration not found in File Store
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	JSON Identifier bad or missing

Error (Validate)

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	Data file or Data Mapping Configuration not found in File Store
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Status Code	Content	Content-Type	Add. Headers	Description
500 *Internal Server Error*	–	–	–	JSON Identifier bad or missing, or Data file or Data Mapping Configuration not found in File Store

Process Data Mapping (PDF/VT to Content Set)

Submits a request to initiate a new Data Mapping operation using a PDF/VT data file specifically.

No Data Mapping configuration or template are specified, and a Content Set will be created based on the default properties extracted from the metadata of the PDF/VT data file.

Request takes no content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/datamining/pdfvtcs/{dataFileId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataFileId	–	–	–	The Managed File ID (or Name) of the PDF/VT data file in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	PDF/VT data file not found in File Store
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Process Data Mapping (PDF/VT to Data Set)

Submits a request to initiate a new Data Mapping operation using a PDF/VT data file specifically.

No Data Mapping configuration is specified, and a Data Set will be created based on the default properties extracted from the metadata of the PDF/VT data file.

Request takes no content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/datamining/pdfvtds/{dataFileId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataFileId	–	–	–	The Managed File ID (or Name) of the PDF/VT data file in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	–	–	–	PDF/VT data file not found in File Store
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Handshake

Queries the availability of the Data Mapping service.

Request

	GET
	/rest/serverengine/workflow/datamining
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: DataMiningRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Data Mapping service.

Request

	GET
	/rest/serverengine/workflow/datamining/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Data Record Entity Service

The following table is a summary of the resources and methods available in the Data Record Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/entity/datarecords	GET
Add Data Records	/rest/serverengine/entity/datarecords	POST
Get Data Record Values	/rest/serverengine/entity/datarecords/{dataRecordId}/values	GET
Update Data Record Values	/rest/serverengine/entity/datarecords/{dataRecordId}/values	PUT
Get Data Record Properties	/rest/serverengine/entity/datarecords/{dataRecordId}/properties	GET
Update Data Record Properties	/rest/serverengine/entity/datarecords/{dataRecordId}/properties	PUT
Get Multiple Data Record Values	/rest/serverengine/entity/datarecords/values	GET
Get Multiple Data Record Values (JSON)	/rest/serverengine/entity/datarecords/values	POST
Update Multiple Data Record Values	/rest/serverengine/entity/datarecords	PUT
Update Multiple Data Record Properties	/rest/serverengine/entity/datarecords/properties	PUT
Service Version	/rest/serverengine/entity/datarecords/version	GET

Add Data Records

Submits a request to add one or more Data Record entities to one or more entities in the Server as either:

- a Data Record of an existing Data Set entity in the Server, or
- a nested Data Record in a Data Table of an existing Data Record entity in the Server

Request takes JSON New Record Lists as content (each with the Data Set/Data Record ID, Data Table and the new records/values), and on success returns a response containing no content.

Request

	POST
	/rest/serverengine/entity/datarecords
	JSON New Record Lists of the new Data Records
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	–	–	–	Data Records for Data Set/Data Record entities successfully added

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	JSON New Record Lists invalid or missing required structure

Get Data Record Properties

Returns a list of the properties for a specific Data Record entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the properties for the Data Record.

Request

	GET
	/rest/serverengine/entity/datarecords/{dataRecordId}/properties
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataRecordId	–	–	–	The ID of the Data Record entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of properties for Data Record	application/json	–	Data Record entity properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Data Record Values

Returns a list of the values for a specific Data Record entity, and potentially the values of any nested Data Records (if recursive).

Request takes no content, and on success returns a response containing either:

- JSON Record Content List of all the values for the Data Record
- JSON Record Content List (Explicit Types) of all the values and data types for the Data Record
- JSON Record Content List (simplified) of all the values for the Data Record

Request

	GET
	/rest/serverengine/entity/datarecords/{dataRecordId}/values
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataRecordId	–	–	–	ID of the Data Record entity in Server

Query

Name	Required	Type	Default Value	Description
recursive	–	–	False	Recurse all Data Tables within the Data Record and retrieve the values of any nested Data Records
explicitTypes	–	–	False	*Deprecated* Retrieve both values and data types of the Data Record
outputFormat	–	–	default	The output format of the data (either `default`, `explicitTypes` or `simplified`).

Note: Query parameters are **case-sensitive**.

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Record Content List of the values for Data Record	application/json	–	Data Record entity values successfully retrieved
200 *OK*	JSON Record Content List (Explicit Types) of the values and data types for Data Record	application/json	–	Data Record entity values and data types successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or invalid Data Record ID specified

Get Multiple Data Record Values

Returns a list of the values for one or more Data Record entities, and potentially the values of any nested Data Records (if recursive).

Request takes no content, and on success returns a response containing either:

- JSON Record Content Lists of all the values for each Data Record.
- JSON Record Content Lists (Explicit Types) of all the values and data types for each Data Record
- JSON Record Content Lists (simplified) of all the values for each Data Record.

Request

	GET
	/rest/serverengine/entity/datarecords/values
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Query

Name	Required	Type	Default Value	Description
id	–	–	–	ID of each Data Record entity in Server
recursive	–	–	False	Recurse all Data Tables within each Data Record and retrieve the values of any nested Data Records
outputFormat	–	–	default	The output format of the data (either `default`, `explicitTypes` or `simplified`).

Note: Query parameters are **case-sensitive**.

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Record Content Lists of the values for each Data Record	application/json	–	Data Record entity values successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or invalid Data Record ID specified

Get Multiple Data Record Values (JSON)

Returns a list of the values for one or more Data Record entities, and potentially the values of any nested Data Records (if recursive).

Request takes a JSON Data Record Identifier List (with Parameters) of the Data Record IDs as content, and on success returns a response, depending on the specified output format, containing either:

- JSON Record Content Lists of all the values for each Data Record (this is the default)
- JSON Record Content Lists (Explicit Types) of all the values and data types for each Data Record
- JSON Record Content Lists (simplified) of all the values for each Data Record

Request

	POST
	/rest/serverengine/entity/datarecords/values
	JSON Data Record Identifier List (with Parameters) specifying the Data Record entity IDs
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Record Content Lists of the values for each Data Record	application/json	–	Data Record entity values successfully retrieved
200 *OK*	JSON Record Content Lists (Explicit Types) of the values and data types for each Data Record	application/json	–	Data Record entity values and data types successfully retrieved
200 *OK*	JSON Record Content Lists (simplified) of the values for each Data Record	application/json		Data Record entity values successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or invalid Data Record ID specified

Service Handshake

Queries the availability of the Data Record Entity service.

Request

	GET
	/rest/serverengine/entity/datarecords
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: DataRecordEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Data Record Entity service.

Request

	GET
	/rest/serverengine/entity/datarecords/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Data Record Properties

Submits a request to update (and replace) the properties for a specific Data Record entity in the Server.

Request takes a JSON Name/Value List as content (the Data Record ID and the new properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/datarecords/{dataRecordId}/properties
	JSON Name/Value List of properties for Data Record
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataRecordId	–	–	–	ID of the Data Record entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Data Record	text/plain	–	Update of Data Record properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Data Record ID mismatch in JSON

Update Data Record Values

Submits a request to update one or more values for a specific Data Record entity in the Server.

Request takes a JSON Record Content List (Fields Only) as content (the Data Record ID and the new values), and on success returns a response containing no content.

Request

	PUT
	/rest/serverengine/entity/datarecords/{dataRecordId}/values
	JSON Record Content List (Fields Only) of the values for Data Record
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataRecordId	–	–	–	ID of the Data Record entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	–	–	–	Data Record entity values successfully updated

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Data Record ID mismatch in JSON

Update Multiple Data Record Properties

Submits a request to update one or more properties for one or more Data Record entities in the Server.

Request takes JSON Name/Value Lists as content (each with the Data Record ID and the new properties), and on success returns a response containing no content.

Request

	PUT
	/rest/serverengine/entity/datarecords/properties
	JSON Name/Value Lists of the properties of the Data Records
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	–	–	–	Properties of Data Record entities successfully updated

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Multiple Data Record Values

Submits a request to update one or more values for one or more Data Record entities in the Server.

Request takes JSON Record Content Lists (Fields Only) as content (each with the Data Record ID and the new values), and on success returns a response containing no content.

Request

	PUT
	/rest/serverengine/entity/datarecords
	JSON Record Content Lists (Fields Only) of the values for the Data Records
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	–	–	–	Values of Data Record entities successfully updated

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Data Set Entity Service

The following table is a summary of the resources and methods available in the Data Set Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
"Service Handshake" below	/rest/serverengine/entity/datasets	GET
Get All Data Sets	/rest/serverengine/entity/datasets	GET
Get Data Records for Data Set	/rest/serverengine/entity/datasets/{dataSetId}	GET
Delete Data Set Entity	/rest/serverengine/entity/datasets/{dataSetId}/delete	POST
Get Data Set Properties	/rest/serverengine/entity/datasets/{dataSetId}/properties	GET
Update Data Set Properties	/rest/serverengine/entity/datasets/{dataSetId}/properties	PUT
Service Version	/rest/serverengine/entity/datasets/version	GET

Service Handshake

Queries the availability of the Data Set Entity service.

Request

	GET
	/rest/serverengine/entity/datasets
	–
	–
	text/plain
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: DataSetEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Delete Data Set Entity

Submits a request for a specific Data Set entity to be marked for deletion from the Server.

Request takes no content, and on success returns a response containing the result of the request for deletion (*"true"* or *"false"*).

Request

	POST
	/rest/serverengine/entity/datasets/{dataSetId}/delete
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataSetId	–	–	–	ID of the Data Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request for Data Set removal	text/plain	–	Deletion of Data Set successfully requested from Server (response of "true" for success or "false" for failure)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get All Data Sets

Returns a list of all the Data Set entities currently contained within the Server.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Data Sets.

Request

	GET
	/rest/serverengine/entity/datasets
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Data Sets in Server	application/json	–	Identifier List of Data Sets returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Data Records for Data Set

Returns a list of all the Data Record entities contained within a specific Data Set entity.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Data Records in the Data Set.

Request

	GET
	/rest/serverengine/entity/datasets/{dataSetId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataSetId	–	–	–	ID of the Data Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Data Records in Data Set	application/json	–	Identifier List of Data Records returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Data Set Properties

Returns a list of the properties for a specific Data Set entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the properties for the Data Set.

Request

	GET
	/rest/serverengine/entity/datasets/{dataSetId}/properties
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataSetId	–	–	–	ID of the Data Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of properties for Data Set	application/json	–	Data Set entity properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Data Set Entity service.

Request

	GET
	/rest/serverengine/entity/datasets/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Data Set Properties

Submits a request to update (and replace) the properties for a specific Data Set entity in the Server.

Request takes a JSON Name/Value List as content (the Data Set ID and the new properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/datasets/{dataSetId}/properties
	JSON Name/Value List of properties for Data Set
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
dataSetId	–	–	–	ID of the Data Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Data Set	text/plain	–	Update of Data Set properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Data Set ID mismatch in JSON

Document Entity Service

The following table is a summary of the resources and methods available in the Document Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/entity/documents	GET
Get Document Metadata Properties	/rest/serverengine/entity/documents/{documentId}/metadata	GET
Update Document Metadata Properties	/rest/serverengine/entity/documents/{documentId}/metadata	PUT
Service Version	/rest/serverengine/entity/documents/version	GET

Get Document Metadata Properties

Returns a list of the metadata properties for a specific Document entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the metadata properties for the Document.

Request

	GET
	/rest/serverengine/entity/documents/{documentId}/metadata
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
documentId	–	–	–	The ID of the Document entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of metadata properties for Document	application/json	–	Document entity metadata properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Handshake

Queries the availability of the Document Entity service.

Request

	GET
	/rest/serverengine/entity/documents
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: DocumentEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Document Entity service.

Request

	GET
	/rest/serverengine/entity/documents/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Document Metadata Properties

Submits a request to update (and replace) the metadata properties for a specific Document entity in the Server.

Request takes a JSON Name/Value List as content (the Document ID and the new metadata properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/documents/{documentId}/metadata
	JSON Name/Value List of metadata properties for Document
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
documentId	–	–	–	The ID of the Document entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Document	text/plain	–	Update of Document metadata properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Document ID mismatch in JSON

Document Set Entity Service

The following table is a summary of the resources and methods available in the Document Set Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/entity/documentsets	GET
Get Documents for Document Set	/rest/serverengine/entity/documentsets/{documentSetId}	GET
Get Document Set Metadata Properties	/rest/serverengine/entity/documentsets/{documentSetId}/metadata	GET
Update Document Set Metadata Properties	/rest/serverengine/entity/documentsets/{documentSetId}/metadata	PUT
Service Version	/rest/serverengine/entity/documentsets/version	GET

Get Document Set Metadata Properties

Returns a list of the metadata properties for a specific Document Set entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the metadata properties for the Document Set.

Request

	GET
	/rest/serverengine/entity/documentsets/{documentSetId}/metadata
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
documentSetId	–	–	–	The ID of the Document Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of metadata properties for Document Set	application/json	–	Document Set entity metadata properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Documents for Document Set

Returns a list of all the Document entities contained within a specific Document Set entity.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Documents in the Document Set.

Request

	GET
	/rest/serverengine/entity/documentsets/{documentSetId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
documentSetId	–	–	–	The ID of the Document Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Documents in Document Set	application/json	–	Identifier List of Documents returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Handshake

Queries the availability of the Document Set Entity service.

Request

	GET
	/rest/serverengine/entity/documentsets
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: DocumentSetEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Document Set Entity service.

Request

	GET
	/rest/serverengine/entity/documentsets/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Document Set Metadata Properties

Submits a request to update (and replace) the metadata properties for a specific Document Set entity in the Server.

Request takes a JSON Name/Value List as content (the Document Set ID and the new metadata properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/documentsets/{documentSetId}/metadata
	JSON Name/Value List of metadata properties for Document Set
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
documentSetId	–	–	–	The ID of the Document Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Document Set	text/plain	–	Update of Document Set metadata properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Document Set ID mismatch in JSON

Entity Service

The following table is a summary of the resources and methods available in the Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/entity	GET
Find Data Entity	/rest/serverengine/entity/find	PUT
Service Version	/rest/serverengine/entity/version	GET

Find Data Entity

Submits data entity search criteria to the PReS Connect Server.

Request takes a JSON Search Parameters structure as content and on success returns a response containing JSON Identifier Lists (with Sort Key) of the data entity IDs matching the search criteria.

	PUT
	/rest/serverengine/entity/find
	JSON Search Parameters containing search criteria/rules
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier Lists (with Sort Key) containing all the entities matching the search criteria	application/json	–	Search request successfully executed

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Invalid JSON structure specified

Service Handshake

Queries the availability of the Entity service.

Request

	GET
	/rest/serverengine/entity
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: EntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Entity service.

Request

	GET
	/rest/serverengine/entity/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

File Store Service

The following table is a summary of the resources and methods available in the File Store service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/filestore	GET
Synchronize/Upload Managed File (Internal Only)	/rest/serverengine/filestore/file/{fileId}	POST
Synchronize/Upload Managed Directory (Internal Only)	/rest/serverengine/filestore/dir/{fileId}	POST
"Managed File or Directory Exists" on page 530	/rest/serverengine/filestore/file/{fileId}	HEAD
Download Managed File or Directory	/rest/serverengine/filestore/file/{fileId}	GET
"Download Contents of Managed Directory" on page 522	/rest/serverengine/filestore/file/{fileId}/{relPath: .+}	GET
Delete Managed File or Directory	/rest/serverengine/filestore/delete/{fileId}	GET
Upload Data File	/rest/serverengine/filestore/DataFile	POST
Upload Data Mapping Configuration	/rest/serverengine/filestore/DataMiningConfig	POST
Upload Template	/rest/serverengine/filestore/template	POST
Upload Job Creation Preset	/rest/serverengine/filestore/JobCreationConfig	POST
Upload Output Creation Preset	/rest/serverengine/filestore/OutputCreationConfig	POST
Service Version	/rest/serverengine/filestore/version	GET

Service Handshake

Queries the availability of the File Store service.

Request

	GET
	/rest/serverengine/filestore
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: FilestoreRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Synchronize/Upload Managed Directory (Internal Only)

Synchronizes/uploads an existing directory of a specific Managed File ID (or Name) from a Server Extension File Store, to the Server's File Store and is used exclusively in a clustered environment.

Request takes zipped file data as content, and on success returns a response containing the Managed File ID (or Name) of the directory.

This method is *strictly* used internally by PReS Connect.

	POST
	/rest/serverengine/filestore/dir/{fileId}
	Directory (as zipped file)
	application/octet-stream
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
fileId	–	–	–	The Managed File ID (or Name) of the file or directory in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID (or Name)	text/plain	–	Directory successfully synchronized/uploaded to File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
405 *Not Allowed*	–	–	–	Directory already exists in File Store

Synchronize/Upload Managed File (Internal Only)

Synchronizes/uploads an existing file of a specific Managed File ID (or Name) from a Server Extension File Store, to the Server's File Store and is used exclusively in a clustered environment.

Request takes binary file data as content, and on success returns a response containing the Managed File ID (or Name) of the file.

This method is *strictly* used internally by PReS Connect.

Request

	POST
	/rest/serverengine/filestore/file/{fileId}
	File
	application/octet-stream
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
fileId	–	–	–	The Managed File ID (or Name) of the file or directory in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID (or Name)	text/plain	–	File successfully synchronized/uploaded to File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
405 *Not Allowed*	–	–	–	File already exists in File Store

Download Managed File or Directory

Obtains an existing file or directory of a specific Managed File ID (or Name) from the File Store.

Request takes no content, and on success returns a response containing the file or directory data (as zipped file).

Request

	GET
	/rest/serverengine/filestore/file/{fileId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
fileId	–	–	–	The Managed File ID (or Name) of the file or directory in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	File	application/octet-stream	**Content-Disposition** – Filename of the Managed File Example: "attachment; filename=Promo-EN.OL-datamapper" **OL-Original-Filename** - The original file name, if available. E.g. "Default Print Settings.OL-jobpreset"	File successfully downloaded from File Store.
200 *OK*	Directory (zipped)	application/zip	**Content-Disposition** – Filename of the Managed Directory Example: "attachment; filename=letter-ol.zip" **OL-Original-Filename** - The original file name, if available. E.g. "Letter.OL-template"	Directory successfully downloaded from File Store.

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	File or directory not found in File Store

Download Contents of Managed Directory

Obtains the contents of a directory of a specific Managed File ID (or Name) from the File Store.

Request takes no content, and on success returns a response containing the file or directory data (as zipped file).

Request

	GET
	/rest/serverengine/filestore/file/{fileId}/{relPath: .+}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
fileId	Yes	String	–	The Managed File ID (or Name) of the file or directory in File Store
relPath	Yes	String	–	The relative path to the specific contents (file or directory) within directory

Query

Name	Required	Type	Default Value	Description
output	No	String	–	The output method to be used for the file

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	File	application/octet-stream	**Content-Disposition** – Filename of the specific file within directory E.g. "attachment; filename=pps-box.jpg" **OL-Original-Filename** - The original file name, if available. E.g. "pps-box.jpg"	File successfully downloaded from File Store

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	File (Base64 Encoded)	application/octet-stream	**Content-Disposition** – File name of the specific file within directory E.g. "attachment; filename=a5ada9e3-dba7-46ef-ab7c-637e82170d56.html" **OL-Original-Filename** - The original file name, if available. E.g. "Webpage-1.html"	File successfully downloaded from File Store. Response when the parameters specified contain an **output** value of base64
200 *OK*	Directory (Zipped)	application/zip	**Content-Disposition** – File name of the specific directory (zipped) within directory E.g. "attachment; filename=Letter.zip" **OL-Original-Filename** - The original file name, if available. E.g. "Letter.OL-template"	Directory successfully downloaded from File Store.

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Directory not found in File Store.

Delete Managed File or Directory

Removes an existing file or directory of a specific Managed File ID (or Name) from the File Store.

Request takes no content, and on success returns a response containing the result of the request for removal.

Request

	GET
	/rest/serverengine/filestore/delete/{fileId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
fileId	–	–	–	The Managed File ID (or Name) of the file or directory in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request for removal	text/plain	–	Removal of file or directory successfully requested from File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*	"JSON Error" on page 35 specifying error message	application/json		Deletion failed / File still required by OL Connect

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Invalid Managed File ID
500 *Internal Server Error*	"JSON Error" on page 35 specifying error message	application/json		Connection refused

Get Report

Returns a report in JSON or XML of a template in the File Store. The report contains the tags specified in the filter, or all tags if no filter was specified.

Request takes no content, and on success returns a response containing a JSON or XML structure that holds information about the Connect template.

Request

	GET
	/filestore/template/report/{fileId}
	–
	–
	• **Accept** (optional) – a comma-separated list of acceptable content types, or */* if the client accepts all content types. Quality values in a weighted list will be ignored. If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
fileId	–	–		The Managed File ID (or Name) of the template in the File Store.

Query

Name	Required	Type	Default Value	Description
filter	–	–		A valid tag name (case sensitive). Examples: sections, media, scripts, datamodel, properties, locale.

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	A JSON or XML structure holding information about a Connect template. The produced JSON is untyped; a value is either a string or an array.	application/json if the client accepts it; otherwise application/xml		Report returned.

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Filter is invalid.
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Template not found in File Store.
406 *Not Acceptable*				None of the content types are supported by the server.

List Resources

Returns a list of one or more types of resources in the File Store: data files, templates, data mapping configurations, job creation presets and output creation presets, depending on the *Type* query parameter. If no *Type* value is passed, all OL Connect resources, but no generic or data files are listed.
Examples:

```
/rest/serverengine/filestore/resources?type=TEMPLATE
/rest/serverengine/filestore/resources?type=TEMPLATE,DATA_MAPPING_CONFIG,JOB_CREATION_CONFIG
```

Request takes no content, and on success returns a response containing a JSON structure that has an array of objects holding base information on Connect resources.

Request

	GET
	/rest/serverengine/filestore/resources
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
type	–	–	If the type parameter is omitted the endpoint will return: • Templates • Data mapping configurations • Job Creation Presets • Output Creation Presets	The type or types (comma-separated) of the files to list. Possible values: • GENERIC: Generic resources, e.g. binary files • TEMPLATE: Templates • DATA_MAPPING_CONFIG: Data mapping configurations • JOB_CREATION_CONFIG: Job creation presets • OUTPUT_CREATION_CONFIG: Output creation presets • DATA_FILE: Data files

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	A JSON structure that has an array of objects holding base information on Connect resources: • id (integer) • name (string) • path (string) • type (string) • size (integer)	application/json		List returned.

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	No resources of the specified type found in File Store.

Managed File or Directory Exists

Checks whether a file or directory of a specific Managed File ID (or Name) exists in the File Store.

Request

	HEAD
	/rest/serverengine/filestore/file/{fileId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
fileId	–	–	–	The Managed File ID (or Name) of the file or directory in the File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*				File or directory exists in the File Store.

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	File or directory not found in File Store

Upload Data File

Submits a data file to the File Store.

Request takes binary file data as content, and on success returns a response containing the new Managed File ID for the data file.

Request

	POST
	/rest/serverengine/filestore/DataFile
	Data File (File)
	application/octet-stream
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Query

Name	Required	Type	Default Value	Description
filename	–	–	–	The file name of the data file to be uploaded
persistent	–	–	false	Whether the data file to be uploaded will be persistent in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID (or Name)	text/plain	–	Data file successfully uploaded to File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Upload Data Mapping Configuration

Submits a Data Mapping configuration to the File Store.

Request takes binary file data as content, and on success returns a response containing the new Managed File ID for the configuration.

Request

	POST
	/rest/serverengine/filestore/DataMiningConfig
	Data Mapping Configuration (File)
	application/octet-stream
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Query

Name	Required	Type	Default Value	Description
filename	–	–	–	The file name of the configuration to be uploaded
persistent	–	–	false	Whether the configuration to be uploaded will be persistent in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID (or Name)	text/plain	–	Configuration successfully uploaded to File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Request contains no data for upload to File Store.
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Upload Template

Submits a template to the File Store.

Request takes zipped file data as content, and on success returns a response containing the new Managed File ID for the template.

Request

	POST
	/rest/serverengine/filestore/template
	Template (File)
	application/zip
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Query

Name	Required	Type	Default Value	Description
filename	–	–	–	The file name of the template to be uploaded
persistent	–	–	false	Whether the template to be uploaded will be persistent in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID (or Name)	text/plain	–	Template successfully uploaded to File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Request contains no data for upload to File Store.
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Upload Job Creation Preset

Submits a Job Creation preset to the File Store.

Request takes XML file data as content, and on success returns a response containing the new Managed File ID for the preset.

Request

	POST
	/rest/serverengine/filestore/JobCreationConfig
	Job Creation Preset (File)
	application/xml
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Query

Name	Required	Type	Default Value	Description
filename	–	–	–	The file name of the preset to be uploaded
persistent	–	–	false	Whether the preset to be uploaded will be persistent in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID (or Name)	text/plain	–	Preset successfully uploaded to File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Request contains no data for upload to File Store.
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Upload Output Creation Preset

Submits an Output Creation preset to the File Store.

Request takes XML file data as content, and on success returns a response containing the new Managed File ID for the preset.

Request

	POST
	/rest/serverengine/filestore/OutputCreationConfig
	Output Creation Preset (File)
	application/xml
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Query

Name	Required	Type	Default Value	Description
filename	–	–	–	The file name of the preset to be uploaded
persistent	–	–	false	Whether the preset to be uploaded will be persistent in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Managed File ID (or Name)	text/plain	–	Preset successfully uploaded to File Store

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
400 *Bad Request*				Request contains no data for upload to File Store.
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the File Store service.

Request

	GET
	/rest/serverengine/filestore/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Job Creation Service

The following table is a summary of the resources and methods available in the Job Creation service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/workflow/jobcreation	GET
Process Job Creation	/rest/serverengine/workflow/jobcreation/{configId}	POST
"Process Job Creation (By Content Set) (JSON)" on page 554	/rest/serverengine/workflow/jobcreation/{configId}	POST
"Process Job Creation (By Content Set) (Runtime Parameters) (JSON)" on page 555	/rest/serverengine/workflow/jobcreation/{configId}	POST
Process Job Creation (By Job Set Structure) (JSON)	/rest/serverengine/workflow/jobcreation	POST
Get All Operations	/rest/serverengine/workflow/jobcreation/getOperations	GET
Get Progress of Operation	/rest/serverengine/workflow/jobcreation/getProgress/{operationId}	GET
Get Result of Operation	/rest/serverengine/workflow/jobcreation/getResult/{operationId}	POST
Cancel an Operation	/rest/serverengine/workflow/jobcreation/cancel/{operationId}	POST
Service Version	/rest/serverengine/workflow/jobcreation/version	GET

Cancel an Operation

Requests the cancellation of a running Job Creation operation of a specific operation ID.

Request takes no content, and on success returns a response with no content.

Request

	POST
	/rest/serverengine/workflow/jobcreation/cancel/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path parameters

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Job Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
204 *No Content*	–	–	–	Operation cancellation requested

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Get All Operations

Returns a list of all the workflow operations actively running on the Server.

Request takes no content, and on success returns a response containing a JSON Operations List of all the actively running operations.

Request

	GET
	/rest/serverengine/workflow/jobcreation/getOperations
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Operations List of all the actively running operations in Server	application/json	–	List of actively running operations successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Progress of Operation

Retrieves the progress of a running Job Creation operation of a specific operation ID.

Request takes no content, and on success returns a response containing the current value of operation progress (values ranging from 0 – 100, followed by the value of 'done' on completion).

Request

	GET
	/rest/serverengine/workflow/jobcreation/getProgress/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Job Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Progress value of Job Creation operation	text/plain	–	Progress of operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Result of Operation

Retrieves the final result of a completed Job Creation operation of a specific operation ID.

Request takes no content, and on success returns a response containing the ID of the Job Set produced.

Request

	POST
	/rest/serverengine/workflow/jobcreation/getResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Job Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Job Set ID	text/plain	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Process Job Creation

Submits a request to initiate a new Job Creation operation.

Request takes no content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/jobcreation/{configId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
configId	–	–	–	The Managed File ID (or Name) of the Job Creation Preset in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Job Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Job Creation Preset not found in File Store

Process Job Creation (By Content Set) (JSON)

Submits a request to initiate a new Job Creation operation.

Request takes a JSON Identifier List of Content Set IDs as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/jobcreation/{configId}
	JSON Identifier List specifying a list of Content Set entity IDs
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path Parameters

Name	Required	Type	Default Value	Description
configId	–	–	–	The Managed File ID (or Name) of the Job Creation Preset in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Job Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Job Creation Preset or Content Set entity not found in File Store/Server

Process Job Creation (By Content Set) (Runtime Parameters) (JSON)

Submits a request to initiate a new Job Creation operation.

Request takes a JSON Identifier List (with Runtime Parameters) of Content Set IDs and parameters as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/jobcreation/{configId}
	JSON Identifier List (with Runtime Parameters) specifying a list of Content Set entity Ids and parameters
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path Parameters

Name	Required	Type	Default Value	Description
configId	Yes	String	–	The Managed File ID (or Name) of the Job Creation Preset in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Job Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	JSON Error specifying error message	application/json	–	Job Creation Preset not found in File Store

Process Job Creation (By Job Set Structure) (JSON)

Submits a request to initiate a new Job Creation operation.

Request takes a JSON Job Set Structure containing a list of Content Items as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/jobcreation
	JSON Job Set Structure describing Job Set (and Content Items)
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Job Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Handshake

Queries the availability of the Job Creation service.

Request

	GET
	/rest/serverengine/workflow/jobcreation
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: JobCreationRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Job Creation service.

Request

	GET
	/rest/serverengine/workflow/jobcreation/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Job Entity Service

The following table is a summary of the resources and methods available in the Job Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/entity/jobs	GET
Get Content Items for Job	/rest/serverengine/entity/jobs/{jobId}/contents	GET
Get Job Segments for Job	/rest/serverengine/entity/jobs/{jobId}	GET
Get Job Metadata Properties	/rest/serverengine/entity/jobs/{jobId}/metadata	GET
Update Job Metadata Properties	/rest/serverengine/entity/jobs/{jobId}/metadata	PUT
Get Job Properties	/rest/serverengine/entity/jobs/{jobId}/properties	GET
Update Job Properties	/rest/serverengine/entity/jobs/{jobId}/properties	PUT
Update Multiple Job Properties	/rest/serverengine/entity/jobs/properties	PUT
Service Version	/rest/serverengine/entity/jobs/version	GET

Get Content Items for Job

Returns a list of all the Content Item entities (and their corresponding Data Record entities) contained within a specific Job entity.

Request takes no content, and on success returns a response containing a JSON Content Item Identifier List of all the Content Items for the Job.

Request

	GET
	/rest/serverengine/entity/jobs/{jobId}/contents
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobId	–	–	–	The ID of the Job entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Content Item Identifier List of all the Content Items in Job	application/json	–	Content Item Identifier List returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Job Metadata Properties

Returns a list of the metadata properties for a specific Job entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the metadata properties for the Job.

Request

	GET
	/rest/serverengine/entity/jobs/{jobId}/metadata
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobId	–	–	–	The ID of the Job entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of metadata properties for Job	application/json	–	Job entity metadata properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Job Properties

Returns a list of the properties for a specific Job entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the properties for the Job.

Request

	GET
	/rest/serverengine/entity/jobs/{jobId}/properties
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobId	–	–	–	The ID of the Job entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of properties for Job	application/json	–	Job entity properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Job Segments for Job

Returns a list of all the Job Segment entities contained within a specific Job entity.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Job Segments in the Job.

Request

	GET
	/rest/serverengine/entity/jobs/{jobId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobId	–	–	–	The ID of the Job entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Job Segments in Job	application/json	–	Identifier List of Job Segments returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Handshake

Queries the availability of the Job Entity service.

Request

	GET
	/rest/serverengine/entity/jobs
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: JobEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Job Entity service.

Request

	GET
	/rest/serverengine/entity/jobs/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Job Metadata Properties

Submits a request to update (and replace) the metadata properties for a specific Job entity in the Server.

Request takes a JSON Name/Value List as content (the Job ID and the new metadata properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/jobs/{jobId}/metadata
	JSON Name/Value List of metadata properties for Job
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobId	–	–	–	The ID of the Job entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Job	text/plain	–	Update of Job metadata properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Job ID mismatch in JSON

Update Job Properties

Submits a request to update (and replace) the properties for a specific Job entity in the Server.

Request takes a JSON Name/Value List as content (the Job ID and the new properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/jobs/{jobId}/properties
	JSON Name/Value List of properties for Job
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobId	–	–	–	The ID of the Job entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Job	text/plain	–	Update of Job properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Job ID mismatch in JSON

Update Multiple Job Properties

Submits a request to update one or more properties for one or more Job entities in the Server.

Request takes JSON Name/Value Lists as content (each with the Job ID and the new properties), and on success returns a response containing no content.

Request

	PUT
	/rest/serverengine/entity/jobs/properties
	JSON Name/Value List of the properties of the Jobs
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	–	–	–	Properties of Job entities successfully updated

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Job Segment Entity Service

The following table is a summary of the resources and methods available in the Job Segment Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/entity/jobsegments	GET
Get Document Sets for Job Segment	/rest/serverengine/entity/jobsegments/{jobSegmentId}	GET
Get Job Segment Metadata Properties	/rest/serverengine/entity/jobsegments/{jobSegmentId}/metadata	GET
Update Job Segment Metadata Properties	/rest/serverengine/entity/jobsegments/{jobSegmentId}/metadata	PUT
Service Version	/rest/serverengine/entity/jobsegments/version	GET

Get Document Sets for Job Segment

Returns a list of all the Document Set entities contained within a specific Job Segment entity.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Document Sets in the Job Segment.

Request

	GET
	/rest/serverengine/entity/jobsegments/{jobSegmentId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSegmentId	–	–	–	The ID of the Job Segment entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Document Sets in Job Segment	application/json	–	Identifier List of Document Sets returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Job Segment Metadata Properties

Returns a list of the metadata properties for a specific Job Segment entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the metadata properties for the Job Segment.

Request

	GET
	/rest/serverengine/entity/jobsegments/{jobSegmentId}/metadata
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSegmentId	–	–	–	The ID of the Job Segment entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of metadata properties for Job Segment	application/json	–	Job Segment entity metadata properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Handshake

Queries the availability of the Job Segment Entity service.

Request

	GET
	/rest/serverengine/entity/jobsegments
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: JobSegmentEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Job Segment Entity service.

Request

	GET
	/rest/serverengine/entity/jobsegments/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Job Segment Metadata Properties

Submits a request to update (and replace) the metadata properties for a specific Job Segment entity in the Server.

Request takes a JSON Name/Value List as content (the Job Segment ID and the new metadata properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/jobsegments/{jobSegmentId}/metadata
	JSON Name/Value List of metadata properties for Job Segment
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSegmentId	–	–	–	The ID of the Job Segment entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Job Segment	text/plain	–	Update of Job Segment metadata properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Job Segment ID mismatch in JSON

Job Set Entity Service

The following table is a summary of the resources and methods available in the Job Set Entity service:

Method Name	Uniform Resource Identifier (URI)	Method Type
"Service Handshake" below	/rest/serverengine/entity/jobsets	GET
Get All Job Sets	/rest/serverengine/entity/jobsets	GET
Get Jobs for Job Set	/rest/serverengine/entity/jobsets/{jobSetId}	GET
Delete Job Set Entity	/rest/serverengine/entity/jobsets/{jobSetId}/delete	POST
Get Job Set Metadata Properties	/rest/serverengine/entity/jobsets/{jobSetId}/metadata	GET
Update Job Set Metadata Properties	/rest/serverengine/entity/jobsets/{jobSetId}/metadata	PUT
Get Job Set Properties	/rest/serverengine/entity/jobsets/{jobSetId}/properties	GET
Update Job Set Properties	/rest/serverengine/entity/jobsets/{jobSetId}/properties	PUT
Service Version	/rest/serverengine/entity/jobsets/version	GET

Service Handshake

Queries the availability of the Job Set Entity service.

Request

	GET
	/rest/serverengine/entity/jobsets
	–
	–
	text/plain
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: JobSetEntityRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Delete Job Set Entity

Submits a request for a specific Job Set entity to be marked for deletion from the Server.

Request takes no content, and on success returns a response containing the result of the request for deletion (*"true"* or *"false"*).

Request

	POST
	/rest/serverengine/entity/jobsets/{jobSetId}/delete
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSetId	–	–	–	The ID of the Job Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request for Job Set removal	text/plain	–	Deletion of Job Set successfully requested from Server (response of "true" for success or "false" for failure)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get All Job Sets

Returns a list of all the Job Set entities currently contained within the Server.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Job Sets.

Request

	GET
	/rest/serverengine/entity/jobsets
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Job Sets in Server	application/json	–	Identifier List of Job Sets returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Job Set Metadata Properties

Returns a list of the metadata properties for a specific Job Set entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the metadata properties for the Job Set.

Request

	GET
	/rest/serverengine/entity/jobsets/{jobSetId}/metadata
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSetId	–	–	–	The ID of the Job Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of metadata properties for Job Set	application/json	–	Job Set entity metadata properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Job Set Properties

Returns a list of the properties for a specific Job Set entity.

Request takes no content, and on success returns a response containing a JSON Name/Value List (Properties Only) of all the properties for the Job Set.

Request

	GET
	/rest/serverengine/entity/jobsets/{jobSetId}/properties
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSetId	–	–	–	The ID of the Job Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Name/Value List (Properties Only) of properties for Job Set	application/json	–	Job Set entity properties successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Jobs for Job Set

Returns a list of all the Job entities contained within a specific Job Set entity.

Request takes no content, and on success returns a response containing a JSON Identifier List of all the Jobs in the Job Set.

Request

	GET
	/rest/serverengine/entity/jobsets/{jobSetId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSetId	–	–	–	The ID of the Job Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Identifier List of all the Jobs in Job Set	application/json	–	Identifier List of Jobs returned

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Service Version

Returns the version of the Job Set Entity service.

Request

	GET
	/rest/serverengine/entity/jobsets/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Update Job Set Metadata Properties

Submits a request to update (and replace) the metadata properties for a specific Job Set entity in the Server.

Request takes a JSON Name/Value List as content (the Job Set ID and the new metadata properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/jobsets/{jobSetId}/metadata
	JSON Name/Value List of metadata properties for Job Set
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSetId	–	–	–	The ID of the Job Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Job Set	text/plain	–	Update of Job Set metadata properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Job Set ID mismatch in JSON

Update Job Set Properties

Submits a request to update (and replace) the properties for a specific Job Set entity in the Server.

Request takes a JSON Name/Value List as content (the Job Set ID and the new properties), and on success returns a response containing the result of the request for update/replacement (*"true"*).

Request

	PUT
	/rest/serverengine/entity/jobsets/{jobSetId}/properties
	JSON Name/Value List of properties for Job Set
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
jobSetId	–	–	–	The ID of the Job Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to update Job Set	text/plain	–	Update of Job Set properties successfully requested (response of "true" for success)

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
500 *Internal Server Error*	–	–	–	Server error or Job Set ID mismatch in JSON

Output Creation Service

The following table is a summary of the resources and methods available in the Output Creation service:

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Handshake	/rest/serverengine/workflow/outputcreation	GET
"Process Output Creation (By Job Set)" on page 606	/rest/serverengine/workflow/outputcreation/{configId}/{jobSetId}	POST
"Process Output Creation (By Job Set) (JSON)" on page 608	/rest/serverengine/workflow/outputcreation/{configId}	POST
Process Output Creation (By Job) (JSON)	/rest/serverengine/workflow/outputcreation/{configId}/jobs	POST
Run +PReS Enhance Workflow Configuration	/rest/serverengine/workflow/outputcreation/execute/{args}/{size}	GET
Get All Operations	/rest/serverengine/workflow/outputcreation/getOperations	GET
Get Progress of Operation	/rest/serverengine/workflow/outputcreation/getProgress/{operationId}	GET
Get Result of Operation	/rest/serverengine/workflow/outputcreation/getResult/{operationId}	POST
Get Result of Operation (as Text)	/rest/serverengine/workflow/outputcreation/getResultTxt/{operationId}	POST
Cancel an Operation	/rest/serverengine/workflow/outputcreation/cancel/{operationId}	POST
Service Version	/rest/serverengine/workflow/outputcreation/version	GET

Service Handshake

Queries the availability of the Output Creation service.

Request

	GET
	/rest/serverengine/workflow/outputcreation
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Handshake message: *Server Engine REST Service available: OutputCreationRestService*	text/plain	–	REST Service available

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Process Output Creation (By Job Set)

Submits a request to initiate a new Output Creation operation.

Request takes no content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/outputcreation/{configId}/{jobSetId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
configId	–	–	–	The Managed File ID (or Name) of the Output Creation Preset in File Store
jobSetId	–	–	–	The ID of the Job Set entity in Server

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Output Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*				Output Creation Preset or Job Set entity not found in File Store/Server

Process Output Creation (By Job Set) (JSON)

Submits a request to initiate a new Output Creation operation.

Request takes a JSON Identifier (with Output Parameters) of the Job Set ID as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/outputcreation/{configId}
	JSON Identifier (with Output Parameters) specifying the Job Set entity's ID
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
configId	–	–	–	The Managed File ID (or Name) of the Output Creation Preset in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Output Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*				Output Creation Preset or Job Set entity not found in File Store/Server
500 *Internal Server Error*				JSON Identifier invalid or missing required structure

Process Output Creation (By Job) (JSON)

Submits a request to initiate a new Output Creation operation.

Request takes a JSON Identifier List (with Output Parameters) of the Job IDs as content, and on success returns a response containing additional headers that specify the ID of the new operation as well as link URLs that can be used to retrieve further information/cancel the operation.

Request

	POST
	/rest/serverengine/workflow/outputcreation/{configId}/jobs
	JSON Identifier List (with Output Parameters) specifying the Job entity IDs
	application/json
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
configId	–	–	–	The Managed File ID (or Name) of the Output Creation Preset in File Store

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
202 *Accepted*	–	–	• **operationId** – Operation ID of new Output Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	Creation of new operation successful

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	–	–	–	Output Creation Preset or Job entity not found in File Store/Server
500 *Internal Server Error*	–	–	–	JSON Identifier List invalid or missing required structure

Run +PReS Enhance Workflow Configuration

Submits a request to run a +PReS Enhance workflow configuration via the Weaver engine directly, using a list of command-line arguments.

Request takes no content, and on success returns a response containing the result of the request to run/execute the workflow configuration.

Request

	GET
	/rest/serverengine/workflow/outputcreation/execute/{args}/{size}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

Name	Required	Type	Default Value	Description
args	–	–	–	A comma delimited, URI encoded list of command-line arguments to use to execute the Weaver engine including[1]: • **-c,<workflow file>** – the path to +PReS Enhance workflow configuration • **-O,<output dir>** – the path to directory to be used for output • **<input file>** – the path(s) to the file(s) to be used as input
size	–	–	–	The estimated page size, used by PReS Connect to determine the job size (which determines the number of speed units to use)

[1] For a list of all the available command-line arguments accepted by the Weaver engine, please reference the +PReS Enhance documentation.

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Result of request to run +PReS Enhance workflow	text/plain	–	+PReS Enhance Workflow Configuration successfully run-/executed

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get All Operations

Returns a list of all the workflow operations actively running on the Server.

Request takes no content, and on success returns a response containing a JSON Operations List of all the actively running operations.

Request

	GET
	/rest/serverengine/workflow/outputcreation/getOperations
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	JSON Operations List of all the actively running operations in Server	application/json	–	List of actively running operations successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.

Status Code	Content	Content-Type	Add. Headers	Description
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Progress of Operation

Retrieves the progress of a running Output Creation operation of a specific operation ID.

Request takes no content, and on success returns a response containing the current value of operation progress (values ranging from 0 – 100, followed by the value of 'done' on completion).

Request

	GET
	/rest/serverengine/workflow/outputcreation/getProgress/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Output Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Progress value of Output Creation operation	text/plain	–	Progress of operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Get Result of Operation

Retrieves the final result of a completed Output Creation operation of a specific operation ID.

Request takes no content, and on success returns a response containing either the absolute paths of the final output files produced (multiple spool files) or the content of a final output file (single spool file).

Request

	POST
	/rest/serverengine/workflow/outputcreation/getResult/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Output Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Absolute Paths of the Output Files or the Output File itself	application/octet-stream	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Get Result of Operation (as Text)

Retrieves the final result of a completed Output Creation operation of a specific operation ID.

Request takes no content, and on success returns a response containing the absolute path or paths of the final output file or files produced (single or multiple spool files respectively).

Request

	POST
	/rest/serverengine/workflow/outputcreation/getResultTxt/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Output Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Absolute Path(s) of the Output File(s)	text/plain	–	Result of completed operation successfully retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Cancel an Operation

Requests the cancellation of a running Output Creation operation of a specific operation ID.

Request takes no content, and on success returns a response with no content.

Request

	POST
	/rest/serverengine/workflow/outputcreation/cancel/{operationId}
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Parameters

Path parameters

The following lists the path parameters accepted by this method:

Name	Required	Type	Default Value	Description
operationId	–	–	–	Operation ID of Output Creation operation

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
204 *No Content*	–	–	–	Operation cancellation requested

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.
404 *Not Found*	"JSON Error" on page 35 specifying error message	application/json	–	Operation not found in Server

Service Version

Returns the version of the Output Creation service.

Request

	GET
	/rest/serverengine/workflow/outputcreation/version
	–
	–
	If server security settings are enabled, then *one* of the following: • **Authorization** – Basic Authentication (Username and Password) credentials (Base64 encoded) • **auth_token** – Authorization Token

Response

Success

The following lists status codes indicative of a successful response:

Status Code	Content	Content-Type	Add. Headers	Description
200 *OK*	Version of Service	text/plain	–	Version of REST Service retrieved

Error

The following lists status codes indicative of a failed or error response:

Status Code	Content	Content-Type	Add. Headers	Description
401 *Unauthorized*	–	–	**WWW-Authenticate** – BASIC (Basic Authentication credentials are accepted)	Server authentication is required. Response when neither basic authentication credentials nor an authorization token have been specified in the request headers.
401 *Unauthorized*	Error message	–	–	Server authentication has failed. Response when the authorization token specified in the request headers has now expired.
403 *Forbidden*	Error message	–	–	Server authentication has failed. Response when either the basic authentication credentials or the authorization token specified in the request headers are invalid.

Copyright Information

Legal Notices and Acknowledgements

This guide uses the following third party components:

- **jQuery Library** Copyright © OpenJS Foundation and other contributors. This is distributed under the terms of the Massachusetts Institute of Technology (MIT) license.
- **QUnit Library** Copyright © OpenJS Foundation and other contributors. This is distributed under the terms of the Massachusetts Institute of Technology (MIT) license.