PlanetPress® Connect

OL™ Software

REST API Cookbook with Working Examples

Version: 2019.2

PlanetPress® Connect

OL™ Software

REST API Cookbook with Working Examples
Version 2019.2
Last Revision: 2021-04-14

Objectif Lune, Inc.
2030 Pie-IX, Suite 500
Montréal, QC, Canada, H1V 2C8

+1 (514) 875-5863
www.objectiflune.com

All trademarks displayed are the property of their respective owners.

PlanetPress.
Connect

Table of Contents

PlanetPress.
Connect

Welcome to the PlanetPress Connect REST API Cookbook

This guide is aimed at technically experienced users who wish to learn and use the REST API available in **PlanetPress Connect** version 2019.2.

The PlanetPress 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 PlanetPress Connect server such as:

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

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

This guide is broken down into three sections:

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

- [Workflow & Workflow Processes](#)
- [Input Files](#)
- [Data Entities](#)
- [Workflow Operations](#)
- [JSON Structures](#)

Workflow & Workflow Processes

The primary workflow in PlanetPress 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 PlanetPress 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 design 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 design 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 the content creation process:

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 + Design Template	Content Set(s)
Content Creation Only	Data Records + Design Template	Content Set(s)
Data Mapping + Content Creation + Job Creation	Data File + Data Mapping Configuration + Design Template	Job Set
Data Mapping + Content Creation + Job Creation	Data File + Data Mapping Configuration + Design Template + Job Creation Preset	Job Set
Content Creation + Job Creation	Data Records + Design Template	Job Set
Content Creation + Job Creation	Data Records + Design Template + Job Creation Preset	Job Set
Data Mapping + Content Creation + Job Creation + Output Creation	Data File + Data Mapping Configuration + Design Template + Output Creation Preset	Print Output
Data Mapping + Content Creation + Job Creation + Output Creation	Data File + Data Mapping Configuration + Design Template + Job Creation Preset + Output Creation Preset	Print Output
Content Creation + Job	Data Records + Design Template + Output Creation Preset	Print Output

Processes	Input Combination	Expected Output
Creation + Output Creation		
Content Creation + Job Creation + Output Creation	Data Records + Design 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 PlanetPress Connect workflow:

Name	Relevant Workflow Process	File Name Examples
Data File	Data Mapping	• Promo-EN-10.csv • Promo-EN-10000.csv • PDFVT-Data.pdf
Data Mapping Configuration	Data Mapping	• Promo-EN.OL-datamapper • Transact-EN.OL-datamapper
Design 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 PlanetPress 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 PlanetPress 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 PlanetPress 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 PlanetPress 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 PlanetPress Connect REST API include the following:

- JSON Identifier
- JSON Identifier List
- JSON Name/Value List (Properties Only)
- JSON Name/Value List
- JSON Name/Value Lists

JSON Identifier

Describes an identifier for a single data entity in PlanetPress 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 PlanetPress 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 (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 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 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 PlanetPress Connect REST API include the following:

- JSON Identifier (Managed File)
- JSON Identifier (with Output Parameters)
- JSON Identifier List (with Output Parameters)
- JSON Record Content List
- JSON Record Content Lists
- JSON Record Content List (Explicit Types)
- JSON Record Content Lists (Explicit Types)
- JSON Record Content List (Fields Only)
- JSON Record Content Lists (Fields Only)
- JSON New Record List
- JSON New Record Lists
- JSON Record Data List
- JSON Content Item Identifier List
- JSON Data Record Identifier
- JSON Data Record Identifier List (with Parameters)
- JSON Identifier List (with Email Parameters)
- JSON Record Data List (with Email Parameters)
- JSON HTML Parameters List
- JSON Job Set Structure
- JSON All-In-One Configuration
- JSON Page Details Summary
- JSON Page Details List
- JSON Data Mapping Validation Result
- JSON Search Parameters
- JSON Identifier Lists (with Sort Key)
- JSON Operations List

JSON Identifier (Managed File)

Describes an identifier or named identifier for a single managed file in PlanetPress 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`)

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 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 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 page of the Technical Overview section for further detail on data set and data record entities.

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 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 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](#) page of the [Technical Overview](#) section for further detail on data set and data record entities.

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:

```json
{
    "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": 1332594000000,
        "DueDate": 1335189600000,
        "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 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 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.

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 (Fields Only)

Describes multiple lists of data field values (as name/value pairs) for a data record, used to update existing data record entities of a specific ID.

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 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.

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",
```

```json
                    "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 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`)

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",
```

```json
                "Gender": "STRING",
                "FirstName": "STRING",
                "LastName": "STRING"
            }
        },
        "fields": {
            "ID": "CU01499303",
            "Gender": "Miss",
            "FirstName": "Dianne",
            "LastName": "Straka"
        }
    }
    ]
}
```

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 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 `explicitTypes` parameter determines if the result returned is either 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` – parameter to specify if both data record values and data types are to be retrieved (*type* of `boolean`)

Example

The following is an example of this structure:

```
{
    "recordids": [ 12345, 23456, 34567 ],
    "recursive": true,
    "explicitTypes": false
}
```

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.

Structure

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

- `identifiers` – an `array` of data record entity identifiers (*type* of `number`)
- `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`)
- `user` – the user name to authenticate with (if using authentication) (*type* of `string`)
- `password` – the password to authenticate with (if using authentication) (*type* of `string`)
- `sender` – the email address to be shown as the sender in the email output (*type* of `string`)
- `senderName` – the name to be shown as the sender in the email output (*type* of `string`)
- `useAuth` – parameter to specify if authentication is to be used with the mail server (*type* of `boolean`)
- `useStartTLS` – parameter to specify if Transport Layer Security (TLS) is to be used when sending emails (*type* of `boolean`)
- `useSender` – parameter to specify if the sender address will be used as the receiver address for all emails in the output (*type* of `boolean`)
- `attachWebPage` – parameter to specify if a single HTML web page (with embedded resources) of the Web context should also be created and attached to the email output (*type* of `boolean`)
- `attachPdfPage` – parameter to specify if a PDF of the Print context should also be created and attached to the email output (*type* of `boolean`)

Example

The following is an example of this structure:

```
{
    "identifiers": [
        12345,
        23456
    ],
    "host": "mail.company.com:587",
    "user": "johns",
```

```
      "password": "password5",
      "sender": "john.smith@company.com",
      "useAuth": true,
      "useStartTLS": false,
      "useSender": true,
      "attachWebPage": true,
      "attachPdfPage": true
}
```

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

Structure

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

- `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`)
- `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`)
- `user` – the user name to authenticate with (if using authentication) (*type* of `string`)
- `password` – the password to authenticate with (if using authentication) (*type* of `string`)
- `sender` – the email address to be shown as the sender in the email output (*type* of `string`)
- `senderName` – the name to be shown as the sender in the email output (*type* of `string`)
- `useAuth` – parameter to specify if authentication is to be used with the mail server (*type* of `boolean`)
- `useStartTLS` – parameter to specify if Transport Layer Security (TLS) is to be used when sending emails (*type* of `boolean`)
- `useSender` – parameter to specify if the sender address will be used as the receiver address for all emails in the output (*type* of `boolean`)
- `attachWebPage` – parameter to specify if a single HTML web page (with embedded resources) of the Web context should also be created and attached to the email output (*type* of `boolean`)
- `attachPdfPage` – parameter to specify if a PDF of the Print context should also be created and attached to the email output (*type* of `boolean`)

Example

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"
            }
        },
        {
            "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"
            }
        }
    ],
    "host": "mail.company.com",
    "user": "johns",
    "password": "password5",
    "sender": "john.smith@company.com",
```

```
    "useAuth": true,
    "useStartTLS": false,
    "useSender": true,
    "attachWebPage": true,
    "attachPdfPage": true
}
```

JSON HTML Parameters List

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 (*value* of either `NONE`, `CSS` or `ALL`) (*type* of `string`)

Example

The following is an example of this structure:

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

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
```

PlanetPress.
Connect

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

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

- `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 design template

- `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)

- `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 parameter 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"
    },
    "contentcreation":
    {
        "config": "letter-ol.OL-template"
    },
    "jobcreation":
    {
        "config": "4567"
    },
    "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
    }
}
```

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": {
```

```json
            "name": "Plain Letter Paper",
            "size": "Letter",
            "width": "8.5in",
            "height": "11in"
        }
    }
  ]
}
```

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 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 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.

Note

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 `attributeLOCATION` 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 design 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"
        }
    ]
}
```

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 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.

> **Note**
>
> 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 design 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"
```

```
        }
    ]
```

Working Examples

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

For help on getting started with the PlanetPress 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: http://www.w3schools.com/html/

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 PlanetPress Connect server then the Using the Working Examples with Server Security page should be read also.

Requirements & Installation

Requirements

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

1. A working installation of PlanetPress Connect
2. Any modern web browser able to display HTML5[1]

> **Warning**
>
> If using Internet Explorer, you may find issues when using the working examples with PlanetPress 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 PlanetPress 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 PlanetPress Connect Server Preferences should avoid issues with running the various examples on Internet Explorer.
>
> **It is recommended that you use a modern web-browser other than Internet Explorer when running the working examples.**

---

[1]Any recent version of Mozilla Firefox, Google Chrome, or Opera with support for HTML5 should be suitable for running the working examples contained in this guide. Versions of Internet Explorer 10+ may also be suitable in some cases.

Installation

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

To locate the source on Windows:

1. Open up **Windows Explorer** and navigate to the PlanetPress 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 PlanetPress 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

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.

```
📂 cookbook
   📂 common
      📂 css
         📄 styles.css
      📁 img
      📂 js
         📄 common.js
      📁 lib
      📂 snippets
         📄 results.html
   📂 examples
      📁 rest-service-a
      📁 rest-service-b
      📂 rest-service-c
         📂 js
            📄 rest-service-c-example01.js
            📄 rest-service-c-example02.js
            📄 rest-service-c-example03.js
         📄 rest-service-c-example01.html
         📄 rest-service-c-example02.html
         📄 rest-service-c-example03.html
```

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

`Username`
 	
`Section Name`
	Name (Text) Value	• ol-admin
		• Section 2

`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:

```
Data Set Entity Service - Get All Data Sets Example

 ┌Inputs────────────────────────────────────────────────────────────────────┐
 │ No Input Required                                              [ Submit ]  │
 └───────────────────────────────────────────────────────────────────────────┘


 ┌Results───────────────────────────────────────────────────────────────────┐
 │ Request Successful                                                         │
 │                                                                            │
 │ Data Set IDs:                                                              │
 │                                                                            │
 │   Plain:                                                                   │
 │       61, 88, 115, 158, 222                                                │
 │                                                                            │
 │   JSON Identifier List:                                                    │
 │       {                                                                    │
 │           "identifiers": [                                                 │
 │               61,                                                          │
 │               88,                                                          │
 │               115,                                                         │
 │               158,                                                         │
 │               222                                                          │
 │           ]                                                                │
 │       }                                                                    │
 │                                                               [ Clear  ]   │
 └───────────────────────────────────────────────────────────────────────────┘
```

> **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.

PlanetPress.
Connect

Using the Working Examples with Server Security

If you have the **Server Security Settings** set to *enabled* in your PlanetPress 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 PlanetPress 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-
3.4.1.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>
```

```html
            <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

```javascript
/* 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

```
Authentication Service - Authenticate/Login to Server Example

Inputs
Username:                                              ol-admin
Password:                                              ••••••

Actions
                                                       Submit


Results
User 'ol-admin' Authenticated Successfully

Authorization Token:
  LvGTwfYDA+3tdwBDdfI+9Ysy7vNlsd2lssEpFZhggiE=
                                                       Clear
```

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 Design 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

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Upload Data File Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.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>
```

```html
            <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

```javascript
/* 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

File Store Service - Upload Data File Example

Inputs
Data File: Browse... Promo-EN-1000.csv

Options
Named: ☐
Persistent: ☑

Actions
                                                              Submit

Results
Request Successful

Data File 'Promo-EN-1000.csv' Uploaded Successfully

Managed File ID:
  52
                                                              Clear

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-
3.4.1.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>
```

```html
            </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

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

        c.setupExample();

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

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

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

```javascript
                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

```
File Store Service - Upload Data Mapping Configuration Example

┌Inputs─────────────────────────────────────────────────────────────────────┐
│ Data Mapping Configuration:                    [Browse...] Promo-EN.OL-datamapper │
└───────────────────────────────────────────────────────────────────────────┘
┌Options────────────────────────────────────────────────────────────────────┐
│ Named:                                                              ☐       │
│ Persistent:                                                         ☑       │
└───────────────────────────────────────────────────────────────────────────┘
┌Actions────────────────────────────────────────────────────────────────────┐
│                                                              [  Submit  ]   │
└───────────────────────────────────────────────────────────────────────────┘

┌Results────────────────────────────────────────────────────────────────────┐
│ Request Successful                                                         │
│                                                                            │
│ Data Mapping Configuration 'Promo-EN.OL-datamapper' Uploaded Successfully  │
│                                                                            │
│ Managed File ID:                                                           │
│   56                                                                       │
│                                                              [  Clear  ]    │
└───────────────────────────────────────────────────────────────────────────┘
```

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 Design Template to the File Store

Problem

You want to upload a design 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 design template to the server via the File Store REST service:

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

Example

HTML5

fs-designtemplate-upload.html

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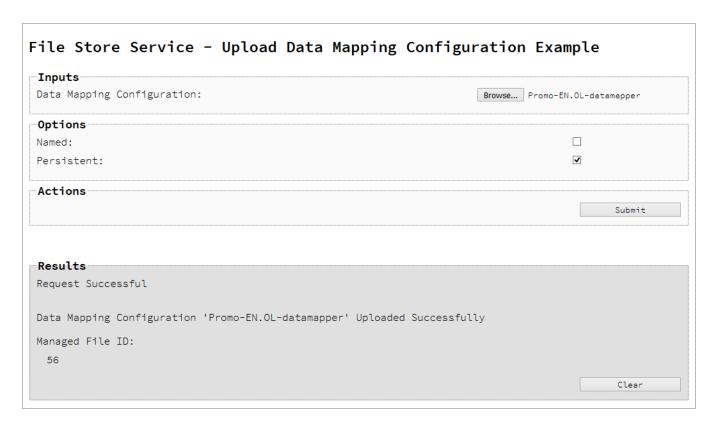
```html
            </div>
        </fieldset>
        <fieldset>
            <legend>Options</legend>
            <div>
                <label for="named">Named:</label>
                <input id="named" type="checkbox" checked>
            </div>
            <div>
                <label for="persistent">Persistent: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

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

        c.setupExample();

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

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

            var file = $("#designtemplate")[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("Design Template '" + file.name +
"' Uploaded Successfully");
                    c.displayResult("Managed File ID", response);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

```
File Store Service - Upload Design Template Example

 Inputs
 Design Template:                              [Browse...] letter-ol.OL-template

 Options
 Named:                                                                    ☑
 Persistent:                                                               ☑

 Actions
                                                               [  Submit  ]


 Results
 Request Successful

 Design Template 'letter-ol.OL-template' Uploaded Successfully

 Managed File ID:
   57

                                                               [  Clear  ]
```

Usage

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

Next you can specify the following options to use with the upload of the design 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 design 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 design template previously selected. This is achieved by getting the first value of the `files` attribute of the HTML element with the ID of `designtemplate` (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 design 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-
3.4.1.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>
```

```html
                </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

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

        c.setupExample();

        $("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");
```
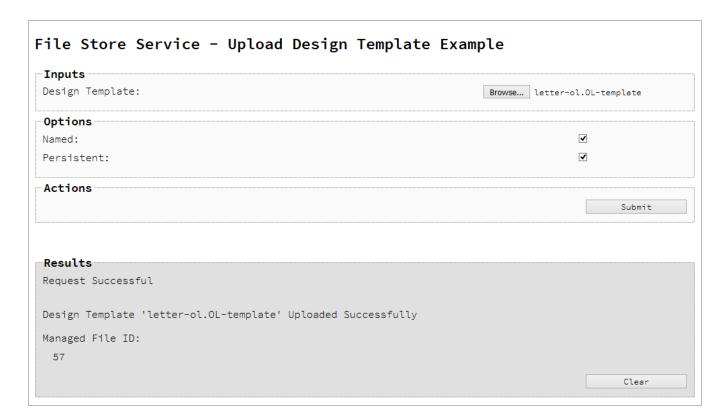
```
            var settings = {
                type:           "POST",
                url:
"/rest/serverengine/filestore/JobCreationConfig?persistent=" +
persistent,
                data:           file,
                processData:    false,
                contentType:    "application/xml"
            };
            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

```
File Store Service - Upload Job Creation Preset Example

┌─Inputs────────────────────────────────────────────────────────────────────────┐
│ Job Creation Preset:                                    Browse...  Promo-EN.OL-jobpreset │
└────────────────────────────────────────────────────────────────────────────────┘
┌─Options───────────────────────────────────────────────────────────────────────┐
│ Named:                                                                    ☐     │
│ Persistent:                                                               ☑     │
└────────────────────────────────────────────────────────────────────────────────┘
┌─Actions───────────────────────────────────────────────────────────────────────┐
│                                                                       Submit    │
└────────────────────────────────────────────────────────────────────────────────┘


┌─Results───────────────────────────────────────────────────────────────────────┐
│ Request Successful                                                              │
│                                                                                 │
│ Job Creation Preset 'Promo-EN.OL-jobpreset' Uploaded Successfully              │
│                                                                                 │
│ Managed File ID:                                                                │
│   58                                                                            │
│                                                                       Clear     │
└────────────────────────────────────────────────────────────────────────────────┘
```

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

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Upload Output Creation Preset Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.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>
```

```html
            </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

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

        c.setupExample();

        $("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

```
File Store Service - Upload Output Creation Preset Example

┌─Inputs──────────────────────────────────────────────────────────────────────┐
│ Output Creation Preset:                      [ Browse... ] Promo-EN.OL-outputpreset │
└─────────────────────────────────────────────────────────────────────────────┘
┌─Options─────────────────────────────────────────────────────────────────────┐
│ Named:                                                              ☐        │
│ Persistent:                                                         ☑        │
└─────────────────────────────────────────────────────────────────────────────┘
┌─Actions─────────────────────────────────────────────────────────────────────┐
│                                                              [   Submit   ]  │
└─────────────────────────────────────────────────────────────────────────────┘

┌─Results─────────────────────────────────────────────────────────────────────┐
│ Request Successful                                                           │
│                                                                              │
│ Output Creation Preset 'Promo-EN.OL-outputpreset' Uploaded Successfully      │
│                                                                              │
│ Managed File ID:                                                             │
│   59                                                                         │
│                                                              [   Clear    ]  │
└─────────────────────────────────────────────────────────────────────────────┘
```

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 PlanetPress 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-
3.2.1.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>
```

```html
                    </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"/>
              </div>
              <div class="option fieldType-number compareType-
condition-">
                    <label for="condition">Condition:</label>
                    <select id="condition" class="options-selector"
data-type="enum-condition-number"/>
              </div>
              <div class="option fieldType-date compareType-
condition-">
                    <label for="condition">Condition:</label>
                    <select id="condition" class="options-selector"
data-type="enum-condition-date"/>
              </div>
              <div class="option fieldType-boolean compareType-
condition-">
                    <label for="condition">Condition:</label>
                    <select id="condition" class="options-selector"
data-type="enum-condition-boolean"/>
              </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-">
```

```html
        <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"/>
            </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-">
```

```html
        <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"/>
            </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"/>
```

```html
            </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"/>
            </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"/>
            </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"/>
            </div>
            <div class="option attribute-STYLE">
                <label for="bindingStyle">Binding Style:</label>
                <select id="bindingStyle" data-type="enum-
bindingstyle"/>
            </div>
            <div class="option attribute-SIDE">
                <label for="bindingEdge">Binding Edge:</label>
                <select id="bindingEdge" data-type="enum-
bindingedge"/>
            </div>
            <div class="option attribute-LOCATION">
                <label for="bindingType">Binding Type:</label>
                <select id="bindingType" data-type="enum-
bindingtype"/>
            </div>
            <div class="option attribute-ANGLE">
                <label for="bindingAngle">Binding Angle:</label>
                <select id="bindingAngle" data-type="enum-
bindingangle"/>
            </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"/>
                </div>
                <div>
                    <label for="value">Value:</label>
                    <input id="value" type="number" step="1"
placeholder="1234" 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"/>
                </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"/>
                </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">
```

```html
            <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>
```

```html
    <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>
```

```html
        <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 {
```

```javascript
                        $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")
```

```javascript
                            .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
```

```javascript
("#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();
                                        }
```

```javascript
                                        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) {
```

```javascript
                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],
            };
```

```javascript
                /* 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; charset=utf-8"
                })
                    .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

Entity Service - Find Data Entity Example

Search Parameters

Entity Type: | Content Items ⌄ |

Search Rules

Rule Type Selector: | Document Size ⌄ |

Rules:

Duplex Rule:
  Condition: | Has Duplex ⌄ |
                                                      [Remove Rule]

Template Rule:
  Condition: | = ⌄ |
  Name: | Rural |
                                                      [Remove Rule]

Rule Set:

  Document Media Rule:
    Attribute: | Name ⌄ |
    Condition: | Contains ⌄ |
    Name: | Impact |
                                                      [Remove Rule]

  Document Media Rule:
    Attribute: | Front Coating ⌄ |
    Condition: | Contains ⌄ |
    Coating: | High Gloss ⌄ |
                                                      [Remove Rule]

  Document Media Rule:
    Attribute: | Back Coating ⌄ |
    Condition: | Contains ⌄ |
    Coating: | Semi Gloss ⌄ |
                                                      [Remove Rule]

  Rules Condition: | Match All Rules ⌄ |
                                                      [Add Search Rule] [Remove Set]

Rule Set:

  Document Binding Rule:
    Attribute: | Style ⌄ |
    Condition: | Contains ⌄ |
    Binding Style: | Stapled ⌄ |
                                                      [Remove Rule]

  Document Binding Rule:
    Attribute: | Side ⌄ |
    Condition: | Contains ⌄ |
    Binding Edge: | Left ⌄ |
                                                      [Remove Rule]

  Document Binding Rule:
    Attribute: | Location ⌄ |
    Condition: | Contains ⌄ |
    Binding Type: | Side ⌄ |
                                                      [Remove Rule]

  Document Binding Rule:
    Attribute: | Angle ⌄ |
    Condition: | Contains ⌄ |
    Binding Angle: | Angle ⌄ |
                                                      [Remove Rule]

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 PlanetPress Connect design 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 design 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 PlanetPress 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.

Warning

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 PlanetPress 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

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get All Data Sets Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.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

```javascript
/* 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

```
Data Set Entity Service - Get All Data Sets Example

┌Inputs────────────────────────────────────────────────────────────────┐
│ No Input Required                                       [   Submit   ] │
│                                                                        │
└────────────────────────────────────────────────────────────────────────┘

┌Results───────────────────────────────────────────────────────────────┐
│ Request Successful                                                     │
│                                                                        │
│ Data Set IDs:                                                          │
│                                                                        │
│   Plain:                                                               │
│       61, 88, 115, 158, 222                                            │
│                                                                        │
│   JSON Identifier List:                                                │
│                                                                        │
│       {                                                                │
│           "identifiers": [                                             │
│               61,                                                      │
│               88,                                                      │
│               115,                                                     │
│               158,                                                     │
│               222                                                      │
│           ]                                                            │
│       }                                                                │
│                                                        [   Clear   ]   │
└────────────────────────────────────────────────────────────────────────┘
```

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

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get Data Records for Data Set Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.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

```
Data Set Entity Service - Get Data Records for Data Set Example

Inputs
Data Set ID:                                              222

Actions
                                                       Submit


Results
Request Successful

Data Record IDs for Data Set '222':

  Plain:
    4505, 4506, 4507, 4508, 4509, 4510, 4511, 4512, 4513, 4514

  JSON Identifier List:
    {
        "identifiers": [
            4505,
            4506,
            4507,
            4508,
            4509,
            4510,
            4511,
            4512,
            4513,
            4514
        ]
    }
                                                       Clear
```

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 PlanetPress 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-
3.4.1.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>Inputs</legend>
                <div>
                    <label for="submit">No Input Required</label>
                    <input id="submit" type="submit"
value="Submit">
                </div>
```

```
                    </fieldset>
            </form>
        </body>
</html>
```

JavaScript/jQuery

cse-get-all-contentsets.js

```javascript
/* 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;

            $.ajax({
                type:   "GET",
                url:    "/rest/serverengine/entity/contentsets"
            })
                .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

```
Content Set Entity Service - Get All Content Sets Example

┌Inputs────────────────────────────────────────────────────────────────────────┐
│ No Input Required                                             ┌─────────────┐ │
│                                                              │   Submit    │ │
│                                                              └─────────────┘ │
└──────────────────────────────────────────────────────────────────────────────┘


┌Results───────────────────────────────────────────────────────────────────────┐
│ Request Successful                                                            │
│                                                                               │
│ Content Set IDs:                                                              │
│                                                                               │
│  Plain:                                                                       │
│     200, 248, 305, 306                                                        │
│                                                                               │
│  JSON Identifier List:                                                        │
│     {                                                                         │
│         "identifiers": [                                                      │
│             200,                                                              │
│             248,                                                              │
│             305,                                                              │
│             306                                                              │
│         ]                                                                     │
│     }                                                                         │
│                                                              ┌─────────────┐ │
│                                                              │    Clear    │ │
│                                                              └─────────────┘ │
└──────────────────────────────────────────────────────────────────────────────┘
```

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

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Get Content Items for Content Set Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.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             4505
        4002             4506
        4003             4507
        4004             4508

  JSON Content Item Identifier List:
      {
          "identifiers": [
              {
                  "item": 4001,
                  "record": 4505
              },
              {
                  "item": 4002,
                  "record": 4506
              },
              {
                  "item": 4003,
                  "record": 4507
              },
              {
                  "item": 4004,
                  "record": 4508
              }
          ]
      }

                                                            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 PlanetPress 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-
3.4.1.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

```javascript
/* 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

```
Job Set Entity Service - Get All Job Sets Example

Inputs
No Input Required                                              Submit

Results
Request Successful

Job Set IDs:

  Plain:
     308, 337, 416, 445

  JSON Identifier List:
     {
         "identifiers": [
             308,
             337,
             416,
             445
         ]
     }

                                                              Clear
```

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-
3.4.1.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

```
Job Set Entity Service - Get Jobs for Job Set Example

 Inputs
 Job Set ID:                                        445

 Actions
                                                              Submit


 Results
 Request Successful

 Job IDs for Job Set '445':

   Plain:
       446

   JSON Identifier List:
       {
           "identifiers": [
               446
           ]
       }

                                                              Clear
```

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
6. Running a Content Creation Operation for Print By Data Record (Using JSON)
7. Running a Content Creation Operation for Email By Data Record (Using JSON)
8. Creating Content for Web By Data Record
9. Creating Content for Web By Data Record (Using JSON)
10. Running a Job Creation Operation (Using JSON)
11. Running an Output Creation Operation
12. Running an Output Creation Operation (Using JSON)
13. Running an Output Creation Operation By Job (Using JSON)
14. Running an All-In-One Operation (Using JSON)

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-
3.4.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dm-process.js"></script>
```

```html
        <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; 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

```javascript
/* 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;
```

```javascript
            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");
```

```javascript
                        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-
3.4.1.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; 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

```javascript
/* 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;
```

```javascript
            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);
```

```javascript
                    $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));
```
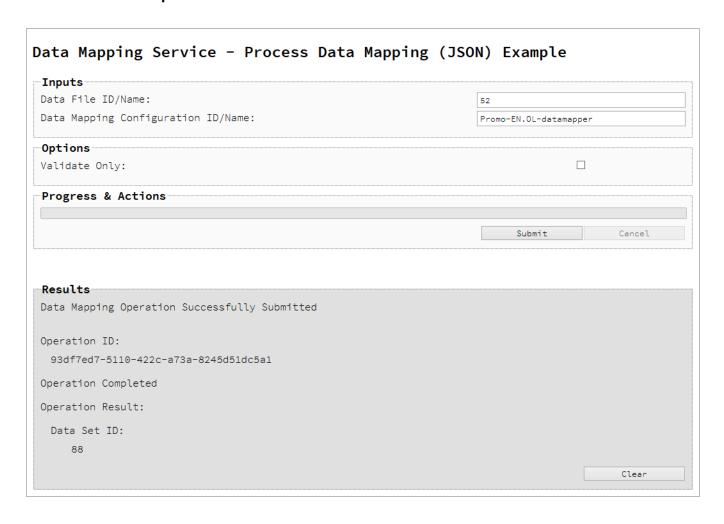
Screenshot & Output

Data Mapping Service - Process Data Mapping (JSON) Example

Inputs

Data File ID/Name: `52`

Data Mapping Configuration ID/Name: `Promo-EN.OL-datamapper`

Options

Validate Only: ☐

Progress & Actions

[   ]

                                                    [Submit]   [Cancel]

Results

Data Mapping Operation Successfully Submitted

Operation ID:

  93df7ed7-5110-422c-a73a-8245d51dc5a1

Operation Completed

Operation Result:

  Data Set ID:

    88

                                                                        [Clear]

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-ds.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Data Mapping (PDF/VT to Data Set)
Example</title>
```

```html
        <script src="../../common/lib/js/jquery-
3.4.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dm-process-pdfvt-ds.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; 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-ds.js

```javascript
/* Data Mapping Service - Process Data Mapping (PDF/VT to Data Set)
Example */
(function ($, c) {
    "use strict";
```

```javascript
$(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) {
```

```javascript
                            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

```
Data Mapping Service - Process Data Mapping (PDF/VT to Data Set)
Example

┌─Inputs──────────────────────────────────────────────────────────────┐
│ Data File ID/Name:                              [54                 ]│
└─────────────────────────────────────────────────────────────────────┘
┌─Progress & Actions──────────────────────────────────────────────────┐
│ [                                                                   ]│
│                                        [  Submit  ]  [  Cancel  ]    │
└─────────────────────────────────────────────────────────────────────┘


┌─Results─────────────────────────────────────────────────────────────┐
│ Data Mapping Operation Successfully Submitted                        │
│                                                                      │
│ Operation ID:                                                        │
│   f07dae50-0670-4f35-9857-7d1225b7bf7c                               │
│                                                                      │
│ Operation Completed                                                  │
│                                                                      │
│ Operation Result:                                                    │
│                                                                      │
│   Data Set ID:                                                       │
│     115                                                              │
│                                        [   Clear   ]                 │
└─────────────────────────────────────────────────────────────────────┘
```

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-cs.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Data Mapping (PDF/VT to Content Set)
Example</title>
```

```html
        <script src="../../common/lib/js/jquery-
3.4.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/dm-process-pdfvt-cs.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; 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>
```
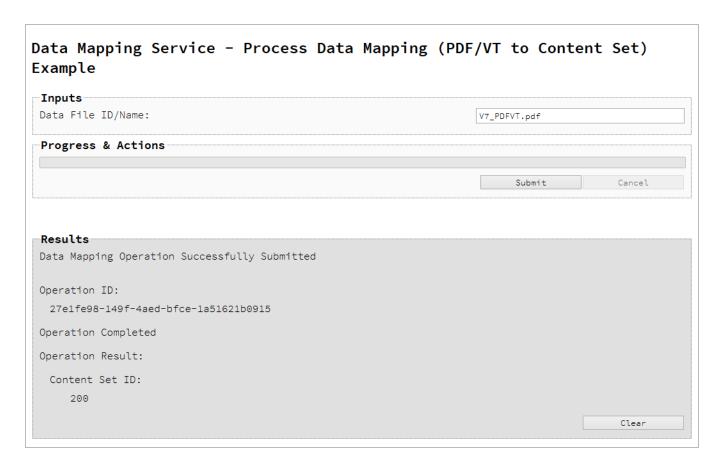
JavaScript/jQuery

dm-process-pdfvt-cs.js

```javascript
/* Data Mapping Service - Process Data Mapping (PDF/VT to Content
Set) Example */
(function ($, c) {
    "use strict";
```

```javascript
$(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

```
Data Mapping Service - Process Data Mapping (PDF/VT to Content Set)
Example

┌─Inputs──────────────────────────────────────────────────────────────────────┐
│ Data File ID/Name:                               V7_PDFVT.pdf                 │
└──────────────────────────────────────────────────────────────────────────────┘
┌─Progress & Actions──────────────────────────────────────────────────────────┐
│ [                                                                          ] │
│                                              Submit            Cancel         │
└──────────────────────────────────────────────────────────────────────────────┘


┌─Results─────────────────────────────────────────────────────────────────────┐
│ Data Mapping Operation Successfully Submitted                                │
│                                                                              │
│ Operation ID:                                                                │
│   27e1fe98-149f-4aed-bfce-1a51621b0915                                        │
│                                                                              │
│ Operation Completed                                                          │
│                                                                              │
│ Operation Result:                                                            │
│                                                                              │
│   Content Set ID:                                                            │
│     200                                                                      │
│                                                                              │
│                                                 Clear                         │
└──────────────────────────────────────────────────────────────────────────────┘
```

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

Problem

You want to run a content creation operation to produce a Content Set using a design 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.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Content Creation Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.min.js"></script>
```

```
        <script src="../../common/js/common.js"></script>
        <script src="js/cc-process.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Content Creation Service - Process Content Creation
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="designtemplate">Design Template
ID/Name:</label>
                    <input id="designtemplate" type="text"
placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &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>
```
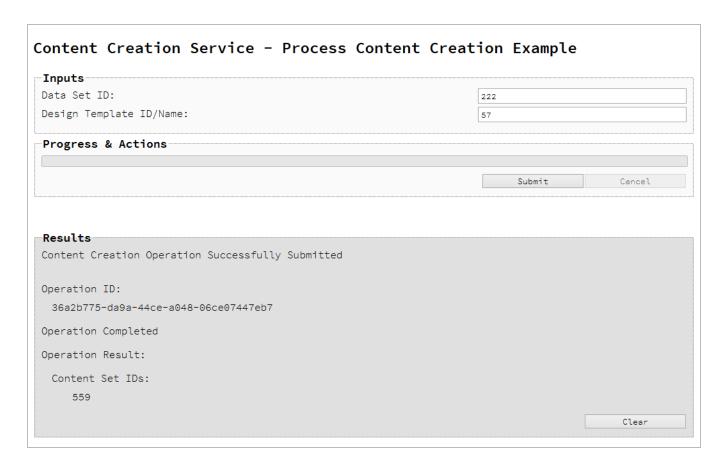
JavaScript/jQuery

cc-process.js

```
/* Content Creation Service - Process Content Creation 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 = $("#designtemplate").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 */
                $.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

```
Content Creation Service - Process Content Creation Example

 ┌Inputs────────────────────────────────────────────────────────────┐
 │ Data Set ID:                                       222            │
 │ Design Template ID/Name:                           57             │
 └──────────────────────────────────────────────────────────────────┘
 ┌Progress & Actions────────────────────────────────────────────────┐
 │ [                                                              ]  │
 │                                    Submit          Cancel        │
 └──────────────────────────────────────────────────────────────────┘


 ┌Results───────────────────────────────────────────────────────────┐
 │ Content Creation Operation Successfully Submitted                │
 │                                                                  │
 │ Operation ID:                                                    │
 │   36a2b775-da9a-44ce-a048-06ce07447eb7                           │
 │                                                                  │
 │ Operation Completed                                              │
 │                                                                  │
 │ Operation Result:                                                │
 │                                                                  │
 │   Content Set IDs:                                               │
 │     559                                                          │
 │                                                                  │
 │                                                    Clear         │
 └──────────────────────────────────────────────────────────────────┘
```

Usage

To run the example simply enter the **Data Set ID** and the **Managed File ID or Name** of your design 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 design 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">
```

```html
        <title>Process Content Creation (By Data Record) (JSON)
Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.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="designtemplate">Design Template
ID/Name:</label>
                    <input id="designtemplate" type="text"
placeholder="1234 or Filename" required>
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress &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>
```
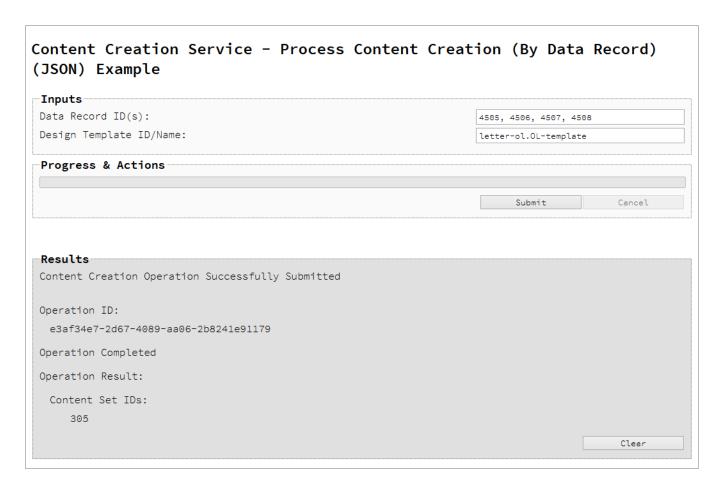
JavaScript/jQuery

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 = $("#designtemplate").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);
```

```javascript
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

```
Content Creation Service - Process Content Creation (By Data Record)
(JSON) Example

┌ Inputs ─────────────────────────────────────────────────────────────┐
  Data Record ID(s):                          │ 4505, 4506, 4507, 4508   │
  Design Template ID/Name:                     │ letter-ol.OL-template    │
└─────────────────────────────────────────────────────────────────────┘

┌ Progress & Actions ─────────────────────────────────────────────────┐
  │                                                                   │
                                            [  Submit  ]   [  Cancel  ]
└─────────────────────────────────────────────────────────────────────┘


┌ Results ────────────────────────────────────────────────────────────┐
  Content Creation Operation Successfully Submitted

  Operation ID:
    e3af34e7-2d67-4089-aa06-2b8241e91179

  Operation Completed

  Operation Result:

    Content Set IDs:
      305

                                                         [  Clear  ]
└─────────────────────────────────────────────────────────────────────┘
```

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 design template (previously uploaded to the file store) into the appropriate text fields, and then select the **Submit** button to start the content creation operation.

PlanetPress.
Connect

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 Email By Data Record (Using JSON)

Problem

You want to run a content creation operation to create and send email content using a design 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

```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-
3.4.1.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="designtemplate">Design Template
ID/Name:</label>
                    <input id="designtemplate" 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>
```

```html
<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" required>
</div>
<div>
    <label for="usesender">Use From as To Email
Address:</label>
    <input id="usesender" type="checkbox" checked>
</div>
<div>
    <label for="attachpdf">Attach PDF Page to
Email:</label>
    <input id="attachpdf" type="checkbox">
</div>
<div>
    <label for="attachweb">Attach Web Page to
Email:</label>
    <input id="attachweb" 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>
```

```html
                    <label for="password">Password:</label>
                    <input id="password" type="password"
placeholder="Password">
                </div>
            </fieldset>
            <fieldset>
                <legend>Progress & 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>
```
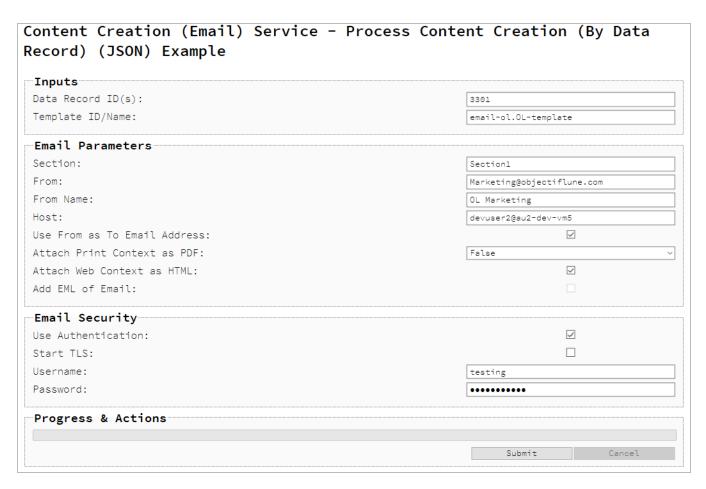
JavaScript/jQuery

cce-process-by-dre-json.js

```javascript
/* Content Creation (Email) Service - Process Content Creation (By
Data Record) (JSON) Example */
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

        var $useAuth = $("#useauth"),
            $startTLS = $("#starttls"),
            $username = $("#username"),
            $password = $("#password"),
            $submitButton = $("#submit"),
            $cancelButton = $("#cancel"),
            $progressBar = $("progress"),
            operationId = null;

        $cancelButton.on("click", function () {
```

```javascript
            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);
            }
        });

        $useAuth.on("click", function (event) {
            var disabled = !($(event.target).prop("checked"));
            $.each([$startTLS, $username, $password], function
(index, $element) {
                $element.prop("disabled", disabled);
            });
        });

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

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

            var dataRecordIds = $("#datarecords").val(),
                templateId = $("#designtemplate").val(),
                section = $("#section").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");
                            c.displaySubResult("Email Report",
response);
                        })
                        .fail(c.displayDefaultFailure);
                };

                /* Construct JSON Identifier List (with Email
Parameters) */
                var config = {
                        "sender":          $("#sender").val(),
                        "host":             $("#host").val(),
                        "useAuth" :         $useAuth.prop("checked"),
                        "useSender":     $("#usesender").prop
("checked"),
                        "attachWebPage":    $("#attachweb").prop
("checked"),
                        "attachPdfPage":    $("#attachpdf").prop
("checked")
                    },
                    senderName = $("#sendername").val().trim(),
                    drids = c.plainIDListToJson(dataRecordIds);

                if (senderName.length) config.senderName = senderName;

                if (config.useAuth) {
                    config.useStartTLS = $startTLS.prop("checked");
                    config.user = $username.val();
                    config.password = $password.val();
                } else {
                    config.user = "";
                }
                config.identifiers = drids.identifiers;

                /* 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; charset=utf-8"
            };
            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 Record) (JSON) Example

Inputs

Data Record ID(s):	3301
Template ID/Name:	email-ol.OL-template

Email Parameters

Section:	Section1
From:	Marketing@objectiflune.com
From Name:	OL Marketing
Host:	devuser2@au2-dev-vm5
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:	testing
Password:	••••••••••

Progress & Actions

Submit   Cancel

Results

Content Creation Operation Successfully Submitted

Operation ID:

  8a1461a2-11ac-4462-96dd-0dfc146cd667

Operation Completed

Operation Result:

  Email Report:

      4 of 4 emails sent

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 design template (previously uploaded to the file store) into the appropriate text fields as your inputs.

Next you need to specify the email parameters to use with the content creation operation:

- **Section** – the section within the Email context of the template to use
- **From** – the email address to be shown as the sender in the email output
- **From Name** – the name to be shown as the sender in the email output
- **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
- **Use From as To Address** – use the sender address as the receiver address for all emails in the output
- **Attach PDF Page to Email** – if a Print context exists in the template, create it's output as a PDF and attach it to the email output
- **Attach Web Page to Email** – if a Web context exists in the template, create it's output as a single HTML web page (with embedded resources) and attach it to email output

Then 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 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, a report of the emails successfully sent will be returned and displayed to the **Results** area.

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 design 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-
3.4.1.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>
```

```html
                    <input id="datarecord" type="text"
placeholder="1234" required>
                </div>
                <div>
                    <label for="designtemplate">Design Template
ID/Name:</label>
                    <input id="designtemplate" 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>
            </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

```javascript
/* 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 = $("#designtemplate").val(),
                section = $("#section").val().trim(),
                params = {
                    inline: $("#inline").val()
                };
            if (section.length) params.section = section;

            /* Process Content Creation (By Data Record) */
            $.ajax({
                type:    "GET",
                url:
"/rest/serverengine/workflow/contentcreation/html/" +
                        templateId + "/" + dataRecordId,
                data:    params
            })
                .done(function (response, status, request) {
                    c.displayHeading("Result");
                    c.displaySubResult("Response",
                        c.htmlToLinkWindow(response, "Result
Link"), false);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

```
Content Creation (HTML) Service - Process Content Creation (By Data
Record) Example

┌─Inputs─────────────────────────────────────────────────────────────┐
│ Data Record ID:                              [4505              ]   │
│ Design Template ID/Name:                     [55                ]   │
└────────────────────────────────────────────────────────────────────┘

┌─HTML Parameters────────────────────────────────────────────────────┐
│ Section:                                     [Section 1         ]   │
│ Inline Mode:                                 [All              ▾]   │
└────────────────────────────────────────────────────────────────────┘

┌─Actions────────────────────────────────────────────────────────────┐
│                                              [    Submit       ]    │
└────────────────────────────────────────────────────────────────────┘


┌─Results────────────────────────────────────────────────────────────┐
│ Result:                                                             │
│                                                                     │
│   Response:                                                         │
│      Result Link                                                    │
│                                              [    Clear        ]    │
└────────────────────────────────────────────────────────────────────┘
```

Usage

To run the example you first need to enter your **Data Record ID** and the **Managed File ID or Name** of your design template (previously uploaded to the file store) into the appropriate text

fields as your inputs.

Next you need to 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

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 design 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-
3.4.1.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>
```

```html
                    <input id="datarecord" type="text"
placeholder="1234" required>
                </div>
                <div>
                    <label for="designtemplate">Design Template
ID/Name:</label>
                    <input id="designtemplate" 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>
            </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

```javascript
/* Content Creation (HTML) Service - Process Content Creation (By
Data Record) (JSON) Example */
```

```javascript
(function ($, c) {
    "use strict";
    $(function () {

        c.setupExample();

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

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

            var dataRecordId = $("#datarecord").val(),
                templateId = $("#designtemplate").val(),
                section = $("#section").val().trim(),
                params = {
                    inline: $("#inline").val()
                };
            if (section.length) params.section = section;

            /* 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; charset=utf-8"
            })
                .done(function (response, status, request) {
                    c.displayHeading("Result");
                    c.displaySubResult("Response",
                        c.htmlToLinkWindow(response, "Result
Link"), false);
                })
                .fail(c.displayDefaultFailure);
        });
    });
}(jQuery, Common));
```

Screenshot & Output

Content Creation (HTML) Service - Process Content Creation (By Data Record) (JSON) Example

Inputs

Data Record ID: | 4506
Design Template ID/Name: | purl-ol-vip-invitation.OL-template

HTML Parameters

Section: | Section 1
Inline Mode: | All

Actions

Submit

Results

Result:

  Response:

    Result Link

Clear

PlanetPress.
Connect

Usage

To run the example you first need to enter your **Data Record ID** and the **Managed File ID or Name** of your design template (previously uploaded to the file store) into the appropriate text

fields as your inputs.

Next you need to 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

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 (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-json.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Job Creation (JSON) Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/jc-process-json.js"></script>
```

```
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Job Creation Service - Process Job Creation (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; 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>
```
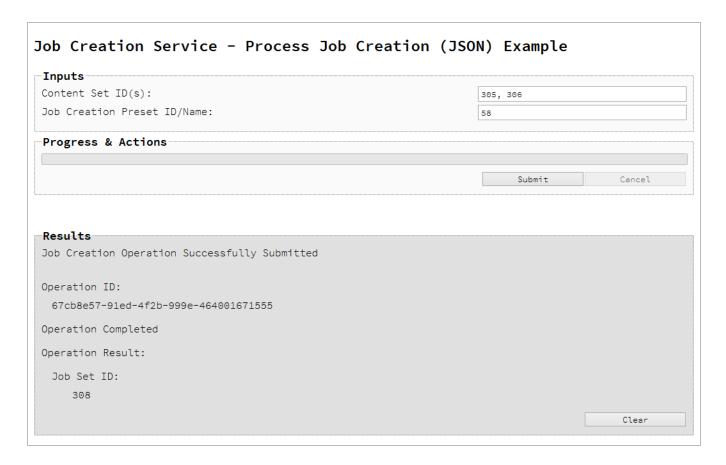
JavaScript/jQuery

jc-process-json.js

```
/* Job Creation Service - Process Job Creation (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 (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

```
Job Creation Service - Process Job Creation (JSON) Example

┌─Inputs─────────────────────────────────────────────────────────────┐
│ Content Set ID(s):                              305, 306            │
│ Job Creation Preset ID/Name:                    58                 │
└────────────────────────────────────────────────────────────────────┘

┌─Progress & Actions─────────────────────────────────────────────────┐
│ [                                                                ] │
│                                      Submit            Cancel      │
└────────────────────────────────────────────────────────────────────┘


┌─Results────────────────────────────────────────────────────────────┐
│ Job Creation Operation Successfully Submitted                      │
│                                                                    │
│ Operation ID:                                                      │
│   67cb8e57-91ed-4f2b-999e-464001671555                             │
│                                                                    │
│ Operation Completed                                                │
│                                                                    │
│ Operation Result:                                                  │
│                                                                    │
│   Job Set ID:                                                      │
│      308                                                           │
│                                                        Clear       │
└────────────────────────────────────────────────────────────────────┘
```

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 an Output Creation Operation

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.html

```
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
```

```
        <title>Process Output Creation Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/oc-process.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Output Creation Service - Process Output Creation
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; 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">
```
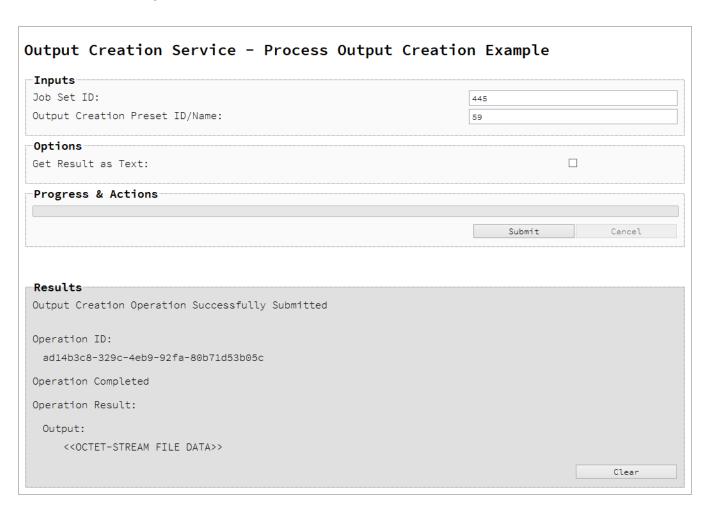
```
            </div>
        </fieldset>
      </form>
    </body>
</html>
```

JavaScript/jQuery

oc-process.js

```
/* Output Creation Service - Process Output Creation 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);
            }
        });
```

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

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

    var jobSetId = $("#jobset").val(),
        configId = $("#ocpreset").val();

    var getFinalResult = function () {

        var result = ($("#resultastxt").prop("checked")) ?
"getResultTxt" : "getResult";

        /* Get Result of Operation */
        $.ajax({
            type:   "POST",
            url:
"/rest/serverengine/workflow/outputcreation/" + result + "/" +
operationId
        })
            .done(function (response, status, request) {
                if (request.getResponseHeader("Content-
Type") === "application/octet-stream")
                    response = "&lt;&lt;OCTET-STREAM FILE
DATA&gt;&gt;";
                c.displayHeading("Operation Result");
                c.displaySubResult("Output", response);
            })
            .fail(c.displayDefaultFailure);
    };

    /* Process Output Creation */
    $.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

Output Creation Service – Process Output Creation Example

Inputs

Job Set ID:	445
Output Creation Preset ID/Name:	59

Options

Get Result as Text: ☐

Progress & Actions

[Submit] [Cancel]

Results

Output Creation Operation Successfully Submitted

Operation ID:
  ad14b3c8-329c-4eb9-92fa-80b71d53b05c

Operation Completed

Operation Result:

  Output:

    <<OCTET-STREAM FILE DATA>>

[Clear]

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 the value *<<OCTET-STREAM FILE DATA>>* 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 (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-json.html

```
<!DOCTYPE html>
<html>
    <head>
```

```html
        <meta charset="utf-8">
        <title>Process Output Creation (JSON) Example</title>
        <script src="../../common/lib/js/jquery-
3.2.1.min.js"></script>
        <script src="../../common/js/common.js"></script>
        <script src="js/oc-process-json.js"></script>
        <link rel="stylesheet" href="../../common/css/styles.css">
    </head>
    <body>
        <h2>Output Creation Service - Process Output Creation
(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; Actions</legend>
                <div>
                    <progress value="0" max="100"></progress>
                </div>
```
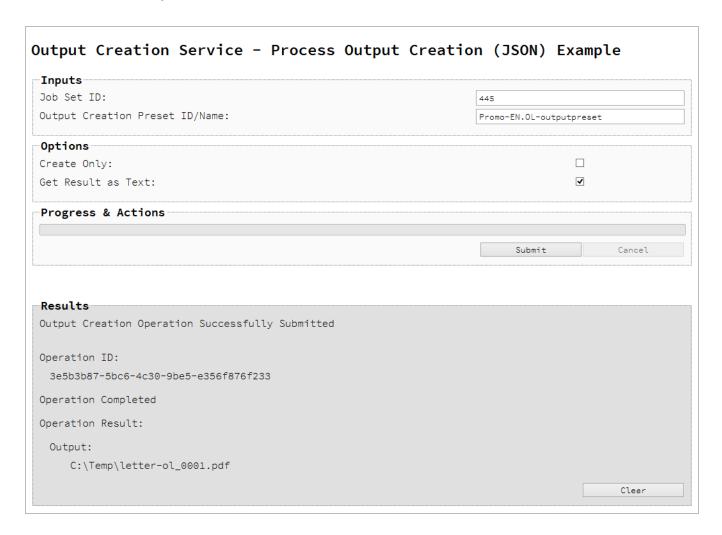
```
                    <div>
                        <input id="cancel" type="button" value="Cancel"
disabled>
                        <input id="submit" type="submit"
value="Submit">
                    </div>
                </fieldset>
            </form>
        </body>
</html>
```

JavaScript/jQuery

oc-process-json.js

```javascript
/* Output Creation Service - Process Output Creation (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);
```

```javascript
                    $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 result = ($("#resultastxt").prop("checked")) ?
"getResultTxt" : "getResult";

            /* Get Result of Operation */
            $.ajax({
                type:   "POST",
                url:
"/rest/serverengine/workflow/outputcreation/" + result + "/" +
operationId
            })
                .done(function (response, status, request) {
                    if (request.getResponseHeader("Content-
Type") === "application/octet-stream")
                        response = "&lt;&lt;OCTET-STREAM FILE
DATA&gt;&gt;";
                    c.displayHeading("Operation Result");
                    c.displaySubResult("Output", response);
                })
                .fail(c.displayDefaultFailure);
        };

        /* Process Output Creation (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

```
Output Creation Service - Process Output Creation (JSON) Example

 Inputs
 Job Set ID:                                            445
 Output Creation Preset ID/Name:                        Promo-EN.OL-outputpreset

 Options
 Create Only:                                                                    ☐
 Get Result as Text:                                                             ☑

 Progress & Actions
 ┌──────────────────────────────────────────────────────────────────────────┐
 │                                                                            │
 └──────────────────────────────────────────────────────────────────────────┘
                                                      Submit        Cancel

 Results
 Output Creation Operation Successfully Submitted

 Operation ID:
   3e5b3b87-5bc6-4c30-9be5-e356f876f233

 Operation Completed

 Operation Result:

   Output:
       C:\Temp\letter-ol_0001.pdf

                                                                      Clear
```

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 the value *<<OCTET-STREAM FILE DATA>>* 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

```html
<!DOCTYPE html>
<html>
    <head>
        <meta charset="utf-8">
        <title>Process Output Creation (By Job) (JSON)
Example</title>
        <script src="../../common/lib/js/jquery-
3.4.1.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; 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>
```
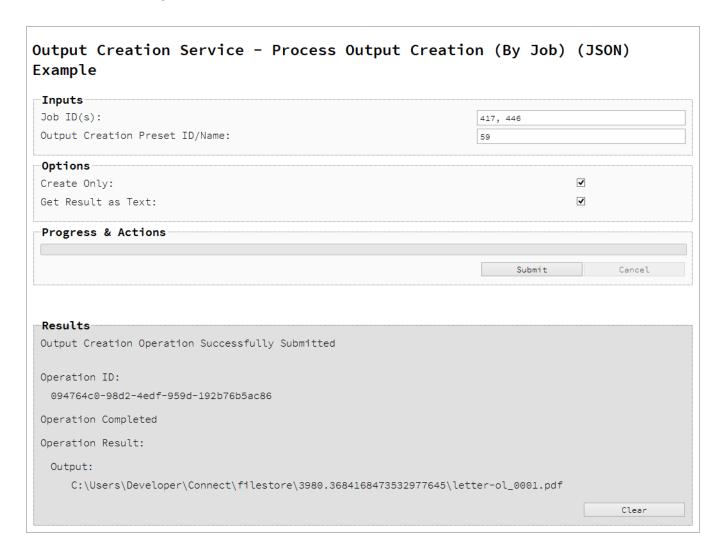
JavaScript/jQuery

oc-process-by-je-json.js

```javascript
/* 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 result = ($("#resultastxt").prop("checked")) ?
"getResultTxt" : "getResult";

                /* Get Result of Operation */
                $.ajax({
                    type:    "POST",
                    url:
"/rest/serverengine/workflow/outputcreation/" + result + "/" +
operationId
                })
                    .done(function (response, status, request) {
                        if (request.getResponseHeader("Content-
Type") === "application/octet-stream")
                            response = "&lt;&lt;OCTET-STREAM FILE
DATA&gt;&gt;";
                        c.displayHeading("Operation Result");
                        c.displaySubResult("Output", response);
                    })
                    .fail(c.displayDefaultFailure);
            };
```

```javascript
                /* 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

```
Output Creation Service - Process Output Creation (By Job) (JSON)
Example
┌─Inputs─────────────────────────────────────────────────────────────────┐
│ Job ID(s):                                    │ 417, 446            │   │
│ Output Creation Preset ID/Name:               │ 59                  │   │
└────────────────────────────────────────────────────────────────────────┘
┌─Options────────────────────────────────────────────────────────────────┐
│ Create Only:                                                    ☑      │
│ Get Result as Text:                                             ☑      │
└────────────────────────────────────────────────────────────────────────┘
┌─Progress & Actions─────────────────────────────────────────────────────┐
│ ┌────────────────────────────────────────────────────────────────────┐ │
│ └────────────────────────────────────────────────────────────────────┘ │
│                                          [  Submit  ]   [  Cancel  ]    │
└────────────────────────────────────────────────────────────────────────┘


┌─Results────────────────────────────────────────────────────────────────┐
│ Output Creation Operation Successfully Submitted                       │
│                                                                        │
│ Operation ID:                                                          │
│   094764c0-98d2-4edf-959d-192b76b5ac86                                 │
│                                                                        │
│ Operation Completed                                                    │
│                                                                        │
│ Operation Result:                                                      │
│                                                                        │
│   Output:                                                              │
│     C:\Users\Developer\Connect\filestore\3980.3684168473532977645\letter-ol_0001.pdf │
│                                                      [  Clear  ]        │
└────────────────────────────────────────────────────────────────────────┘
```

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 the value *<<OCTET-STREAM FILE DATA>>* 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-
```

```
3.4.1.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 & 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-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 =      $("#designtemplate"),
        $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");

    $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);
        }
    });
```

```javascript
        /**
         * @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 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 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");
            $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);
            }

            /* 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;
```

```javascript
                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 result = ($resultastxt.prop("checked")) ?
"getResultTxt" : "getResult";

                        /* Get Result of Operation */
                        $.ajax({
                            type:   "POST",
                            url:
"/rest/serverengine/workflow/print/" + result + "/" + operationId
                        })
                            .done(function (response, status, request)
{
                                if (request.getResponseHeader("Content-
Type") === "application/octet-stream")
                                    response = "<<OCTET-STREAM FILE
DATA>>";

                                c.displayHeading("Operation Result");
                                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`
Design Template ID/Name: `letter-ol.OL-template`

Job Creation

Job Creation Preset ID/Name: `58`

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"
        },
        "outputcreation": {
            "config": "59",
            "createOnly": false
        },
        "printRange": {
            "printRange": "1-5, 7, 9"
```

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 or Name**:

- Data file
- Data Mapping configuration
- Design template
- Job Creation preset
- Output Creation preset

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* processes 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 the value *<<OCTET-STREAM FILE DATA>>* 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 PlanetPress 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 PlanetPress Connect REST API:

Service Name	Internal Name	Description
[Authentication Service](#)	*AuthenticationRestService*	This service exposes methods concerned with server security and authentication with the PlanetPress 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

Service Name	Internal Name	Description
		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

Service Name	Internal Name	Description
		• 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
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

Service Name	Internal Name	Description
		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](#)	*DocumentEntityRestService*	This service exposes methods

Service Name	Internal Name	Description
Service		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

Service Name	Internal Name	Description
		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
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 PlanetPress Connect server. It includes methods to facilitate the following functions: • Uploading of data files and data mapping configurations to the file store

Service Name	Internal Name	Description
		• Uploading of design 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 design template only (no input data) • Getting specific resources contained within a design 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:

Service Name	Internal Name	Description
		- 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:

Service Name	Internal Name	Description
		• Getting the document set IDs contained within a job segment • Getting and updating of job segment metadata properties
[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

Service Name	Internal Name	Description
		• 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

PlanetPress.
Connect

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	/authentication	GET
Authenticate/Login to Server	/authentication/login	POST
Service Version	/authentication/version	GET

Service Handshake

Queries the availability of the Authentication service.

Type:	*GET*		
URI:	/rest/serverengine/authentication		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Handshake message: *Server Engine REST Service available: AuthenticationRestService*	
	Content Type:	text/plain	
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

Authenticate/Login to Server

Submits an authentication request (using credentials) to the PlanetPress 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 (basic user name & password). On success, the response with 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.

> **Warning**
>
> 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 no authorization token and no Authorization header, then the response will come back as *Unauthorized* and will contain an additional **WWW-Authenticate** response header.

Type:	*POST*		
URI:	/rest/serverengine/authentication/login		
Parameters:	–		
Request:	**Add. Headers:**	**Authorization** – Basic User name & Password credentials (Base64 encoded)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	**WWW-Authenticate** – BASIC (Prompt for Basic Authorization Credentials when no Authorization header specified)	

Content:	Authorization Token
Content Type:	text/plain
Status:	200 OK – Server authentication successful, new token generated 401 Unauthorized – Server authentication has failed or no credentials have been provided/specified in request header

Service Version

Returns the version of the Authentication service.

Type:	*GET*		
URI:	/rest/serverengine/authentication/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	200 OK – Version of REST Service retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired	

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	/workflow/contentcreation	GET
Process Content Creation	/workflow/contentcreation/{templateId}/ {dataSetId}	POST
Process Content Creation (By Data Record) (JSON)	/workflow/contentcreation/{templateId}	POST
Process Content Creation (By Data) (JSON)	/workflow/contentcreation/{templateId}	POST
Create Preview PDF	/workflow/contentcreation/pdfpreview/ {templateId}/{dmConfigId}	POST
Create Preview PDF (By Data Record)	/workflow/contentcreation/pdfpreviewdirect	GET
Create Preview PDF (By Data) (JSON)	/workflow/contentcreation/pdfpreview/ {templateId}	POST
Get All Operations	/workflow/contentcreation/getOperations	GET
Get Progress of Operation	/workflow/contentcreation/getProgress/ {operationId}	GET
Get Result of Operation	/workflow/contentcreation/getResult/ {operationId}	POST
Cancel an Operation	/workflow/contentcreation/cancel/ {operationId}	POST

Method Name	Uniform Resource Identifier (URI)	Method Type
Service Version	/workflow/contentcreation/version	GET

Service Handshake

Queries the availability of the Content Creation service.

Type:	*GET*
URI:	/rest/serverengine/workflow/contentcreation
Parameters:	–
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Handshake message: *Server Engine REST Service available: ContentCreationRestService*</td></tr><tr><td>**Content Type:**</td><td>text/plain</td></tr><tr><td>**Status:**</td><td> 200 OK – REST Service available 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired </td></tr></table>

Process Content Creation

Submits a request to initiate a new Content 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.

Type:	*POST*
URI:	/rest/serverengine/workflow/contentcreation/{templateId}/{dataSetId}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store • **dataSetId** – the ID of the Data Set entity in Server

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	Add. Headers:	• **operationId** – Operation ID of new Content Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation
	Content:	–
	Content	–

Type:	
Status:	202 Accepted – Creation of new operation successful 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Design 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 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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/contentcreation/{templateId}		
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Identifier List specifying a list of Data Record entity IDs	
	Content Type:	application/json	
Response:	Add. Headers:	• **operationId** – Operation ID of new Content Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	
	Content:	–	
	Content	–	

Type:	
Status:	• 202 Accepted – Creation of new operation successful
	• 401 Unauthorized – Server authentication required
	• 403 Forbidden – Server authentication has failed or expired
	• 404 Not Found – Design 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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/contentcreation/{templateId}		
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Record Data List specifying a list of data values for the Data Record(s)	
	Content Type:	application/json	
Response:	Add. Headers:	• **operationId** – Operation ID of new Content Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	
	Content:	–	

Content Type:	–
Status:	• 202 Accepted – Creation of new operation successful • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 404 Not Found – Design template not found in File Store

Create Preview PDF

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.

Type:	*POST*
URI:	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}/{dmConfigId}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store • **dmConfigId** – the Managed File ID (or Name) of the Data Mapping configuration in File Store Query: • **persist** – whether the Data Record produced will be persisted in Server (Default Value: true)
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>Data File (File)</td></tr><tr><td>**Content Type:**</td><td>application/octet-stream</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr></table>

Content:	Managed File ID
Content Type:	text/plain
Status:	200 OK – Creation of preview PDF in File Store successful 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Design template, Data Mapping configuration or Data Record entity not found in File Store/Server

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.

Type:	*GET*		
URI:	/rest/serverengine/workflow/contentcreation/pdfpreviewdirect		
Parameters:	Query: • **templateId** – the Managed File ID (or Name) of the design template in File Store • **dataRecordId** – the ID of the Data Record entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Managed File ID	
	Content Type:	text/plain	
	Status:	• 200 OK – Creation of preview PDF in File Store successful	

		401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Design template 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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/contentcreation/pdfpreview/{templateId}		
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Record Data List specifying a list of data values for the Data Record	
	Content Type:	application/json	
Response:	**Add. Headers:**	–	
	Content:	Managed File ID	
	Content Type:	text/plain	
	Status:	• 200 OK – Creation of preview PDF in File	

		Store successful
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 404 Not Found – Design template 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.

Type:	GET		
URI:	/rest/serverengine/workflow/contentcreation/getOperations		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	JSON Operations List of all the actively running operations in Server	
	Content Type:	application/json	
	Status:	• 200 OK – List of actively running operations successfully retrieved • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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).

Type:	*GET*		
URI:	/rest/serverengine/workflow/contentcreation/getProgress/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Content Creation operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Progress value of Content Creation operation	
	Content Type:	text/plain	
	Status:	• 200 OK – Progress of operation successfully retrieved • 401 Unauthorized – Server authentication	

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/contentcreation/getResult/{operationId}	
Parameters:	Path: • **operationId** – Operation ID of Content Creation operation	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Content Set IDs
	Content Type:	text/plain
	Status:	• 200 OK – Result of completed operation successfully retrieved • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*
URI:	/rest/serverengine/workflow/contentcreation/cancel/{operationId}
Parameters:	Path: • **operationId** – Operation ID of Content Creation operation

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	Add. Headers:	–
	Content:	–
	Content Type:	–
	Status:	• 204 No Content – Operation cancellation requested • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Content Creation service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/contentcreation/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	200 OK – Version of REST Service retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired	

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	/entity/contentitems	GET
Get Data Record for Content Item	/entity/contentitems/ {contentItemId}/datarecord	GET
Get Content Item Properties	/entity/contentitems/ {contentItemId}/properties	GET
Update Content Item Properties	/entity/contentitems/ {contentItemId}/properties	PUT
Update Multiple Content Item Properties	/entity/contentitems/properties	PUT
Service Version	/entity/contentitems/version	GET

Service Handshake

Queries the availability of the Content Item Entity service.

Type:	*GET*	
URI:	/rest/serverengine/entity/contentitems	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: ContentItemEntityRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	GET
URI:	/rest/serverengine/entity/contentitems/{contentItemId}/datarecord
Parameters:	Path: • **contentItemId** – the ID of the Content Item entity in Server

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	Add. Headers:	–
	Content:	JSON Data Record Identifier for the Data Record of Content Item
	Content Type:	application/json
	Status:	• 200 OK – Data Record Identifier returned • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*		
URI:	/rest/serverengine/entity/contentitems/{contentItemId}/properties		
Parameters:	Path: • **contentItemId** – the ID of the Content Item entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Name/Value List (Properties Only) of properties for Content Item	
	Content Type:	application/json	
	Status:	• 200 OK – Content Item entity properties successfully retrieved • 401 Unauthorized – Server authentication	

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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"*).

Type:	*PUT*	
URI:	/rest/serverengine/entity/contentitems/{contentItemId}/properties	
Parameters:	Path: ● **contentItemId** – the ID of the Content Item entity in Server	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of properties for Content Item
	Content Type:	application/json
Response:	Add. Headers:	–
	Content:	Result of request to update Content Item
	Content Type:	text/plain
	Status:	● 200 OK – Update of Content Item properties successfully requested (response of *"true"* for

		success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 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.

Type:	*PUT*		
URI:	/rest/serverengine/entity/contentitems/properties		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Name/Value Lists of the properties of the Content Items	
	Content Type:	application/json	
Response:	Add. Headers:	–	
	Content:	–	
	Content Type:	–	
	Status:	• 200 OK – Properties of Content Item entities successfully updated • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Content Item Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/contentitems/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

PlanetPress.
Connect

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
Get All Content Sets	/entity/contentsets	GET
Get Content Items for Content Set	/entity/contentsets/{contentSetId}	GET
Get Page Details for Content Set	/entity/contentsets/{contentSetId}/pages	GET
Delete Content Set Entity	/entity/contentsets/{contentSetId}/delete	POST
Get Content Set Properties	/entity/contentsets/ {contentSetId}/properties	GET
Update Content Set Properties	/entity/contentsets/ {contentSetId}/properties	PUT
Service Version	/entity/contentsets/version	GET

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.

Type:	*GET*	
URI:	/rest/serverengine/entity/contentsets	
Parameters:	–	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	Add. Headers:	–
	Content:	JSON Identifier List of all the Content Sets in Server
	Content Type:	application/json
	Status:	• 200 OK – Identifier List of Content Sets returned • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*		
URI:	/rest/serverengine/entity/contentsets/{contentSetId}		
Parameters:	Path: • **contentSetId** – the ID of the Content Set entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Content Item Identifier List of all the Content Items in Content Set	
	Content Type:	application/json	
	Status:	• 200 OK – Content Item Identifier List returned • 401 Unauthorized – Server authentication	

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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:

- a JSON Page Details Summary of the page details for the Content Set, or
- a JSON Page Details List of the page details for each Content Item in the Content Set

Type:	*GET*		
URI:	/rest/serverengine/entity/contentsets/{contentSetId}/pages		
Parameters:	Path: • **contentSetId** – the ID of the Content Set entity in Server Query: • **detail** – Return a list of details for each Content Item in the Content Set instead of a summary (Default Value: false)		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Page Details Summary containing page	

		details for Content Set
	Content Type:	application/json
	Status:	• 200 OK – Content Set entity page details successfully retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired
Response (Detail):	**Add. Headers:**	–
	Content:	JSON Page Details List containing page details for each Content Item in Content Set
	Content Type:	application/json
	Status:	• 200 OK – Content Set entity page details successfully retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*POST*	
URI:	/rest/serverengine/entity/contentsets/{contentSetId}/delete	
Parameters:	Path: • **contentSetId** – the ID of the Content Set entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Result of request for Content Set removal
	Content Type:	text/plain
	Status:	• 200 OK – Deletion of Content Set successfully requested from Server (response of *"true"* for success or *"false"* for failure) • 401 Unauthorized – Server authentication

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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.

<table>
<tr><td>Type:</td><td colspan="2">GET</td></tr>
<tr><td>URI:</td><td colspan="2"><u>/rest/serverengine/entity/contentsets/{contentSetId}/properties</u></td></tr>
<tr><td>Parameters:</td><td colspan="2">Path:

• contentSetId – the ID of the Content Set entity in Server</td></tr>
<tr><td rowspan="3">Request:</td><td>Add. Headers:</td><td>auth_token – Authorization Token (if server security settings enabled)</td></tr>
<tr><td>Content:</td><td>–</td></tr>
<tr><td>Content Type:</td><td>–</td></tr>
<tr><td rowspan="4">Response:</td><td>Add. Headers:</td><td>–</td></tr>
<tr><td>Content:</td><td>JSON Name/Value List (Properties Only) of properties for Content Set</td></tr>
<tr><td>Content Type:</td><td>application/json</td></tr>
<tr><td>Status:</td><td>• 200 OK – Content Set entity properties successfully retrieved
• 401 Unauthorized – Server authentication</td></tr>
</table>

		required
		- 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*PUT*	
URI:	/rest/serverengine/entity/contentsets/{contentSetId}/properties	
Parameters:	Path: • **contentSetId** – the ID of the Content Set entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of properties for Content Set
	Content Type:	application/json
Response:	**Add. Headers:**	–
	Content:	Result of request to update Content Set
	Content Type:	text/plain
	Status:	• 200 OK – Update of Content Set properties successfully requested (response of *"true"* for

		success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Content Set ID mismatch in JSON

Service Version

Returns the version of the Content Set Entity service.

<table>
<tr>
<td>Type:</td>
<td colspan="2">GET</td>
</tr>
<tr>
<td>URI:</td>
<td colspan="2"><u>/rest/serverengine/entity/contentsets/version</u></td>
</tr>
<tr>
<td>Parameters:</td>
<td colspan="2">–</td>
</tr>
<tr>
<td rowspan="3">Request:</td>
<td>Add. Headers:</td>
<td>auth_token – Authorization Token (if server security settings enabled)</td>
</tr>
<tr>
<td>Content:</td>
<td>–</td>
</tr>
<tr>
<td>Content Type:</td>
<td>–</td>
</tr>
<tr>
<td rowspan="4">Response:</td>
<td>Add. Headers:</td>
<td>–</td>
</tr>
<tr>
<td>Content:</td>
<td>Version of Service</td>
</tr>
<tr>
<td>Content Type:</td>
<td>text/plain</td>
</tr>
<tr>
<td>Status:</td>
<td>

- 200 OK – Version of REST Service retrieved
- 401 Unauthorized – Server authentication required
- 403 Forbidden – Server authentication has failed or expired

</td>
</tr>
</table>

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	/entity/datarecords	GET
Add Data Records	/entity/datarecords	POST
Get Data Record Values	/entity/datarecords/ {dataRecordId}/values	GET
Update Data Record Values	/entity/datarecords/ {dataRecordId}/values	PUT
Get Data Record Properties	/entity/datarecords/ {dataRecordId}/properties	GET
Update Data Record Properties	/entity/datarecords/ {dataRecordId}/properties	PUT
Get Multiple Data Record Values	/entity/datarecords/values	GET
Get Multiple Data Record Values (JSON)	/entity/datarecords/values	POST
Update Multiple Data Record Values	/entity/datarecords	PUT
Update Multiple Data Record Properties	/entity/datarecords/properties	PUT
Service Version	/entity/datarecords/version	GET

Service Handshake

Queries the availability of the Data Record Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/datarecords		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Handshake message: *Server Engine REST Service available: DataRecordEntityRestService*	
	Content Type:	text/plain	
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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.

Type:	*POST*	
URI:	/rest/serverengine/entity/datarecords	
Parameters:	–	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON New Record Lists of the new Data Records
	Content Type:	application/json
Response:	Add. Headers:	–
	Content:	–
	Content Type:	–
	Status:	• 200 OK – Data Records for Data Set/Data

		Record entities successfully added
		- 401 Unauthorized – Server authentication required
		- 403 Forbidden – Server authentication has failed or expired
		- 500 Internal Server Error – JSON New Record Lists invalid or missing required structure

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:

- a JSON Record Content List of all the values for the Data Record, or
- a JSON Record Content List (Explicit Types) of all the values and data types for the Data Record

Type:	*GET*
URI:	/rest/serverengine/entity/datarecords/{dataRecordId}/values
Parameters:	Path: • **dataRecordId** – the ID of the Data Record entity in Server Query: • **recursive** – recurse all Data Tables within the Data Record and retrieve the values of any nested Data Records also (Default Value: false) • **explicitTypes** – retrieve both values and data types of the Data Record (Default Value: false)

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	Add. Headers:	–
	Content:	JSON Record Content List of the values for Data Record
	Content Type:	application/json
	Status:	• 200 OK – Data Record entity values successfully retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 500 Internal Server Error – Server error or invalid Data Record ID specified
Response (Explicit Types):	Add. Headers:	–
	Content:	JSON Record Content List (Explicit Types) of the values and data types for Data Record
	Content Type:	application/json
	Status:	• 200 OK – Data Record entity values and data types successfully retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 500 Internal Server Error – Server error or

		invalid Data Record ID specified

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.

Type:	*PUT*
URI:	/rest/serverengine/entity/datarecords/{dataRecordId}/values
Parameters:	Path: • **dataRecordId** – the ID of the Data Record entity in Server

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Record Content List (Fields Only) of the values for Data Record
	Content Type:	application/json

Response:	**Add. Headers:**	–
	Content:	–
	Content Type:	–
	Status:	• 200 OK – Data Record entity values successfully updated • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Data Record ID mismatch in JSON

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.

Type:	*GET*		
URI:	/rest/serverengine/entity/datarecords/{dataRecordId}/properties		
Parameters:	Path: • **dataRecordId** – the ID of the Data Record entity in Server		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	JSON Name/Value List (Properties Only) of properties for Data Record	
	Content Type:	application/json	
	Status:	• 200 OK – Data Record entity properties successfully retrieved • 401 Unauthorized – Server authentication	

		required
		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*PUT*	
URI:	/rest/serverengine/entity/datarecords/{dataRecordId}/properties	
Parameters:	Path: • **dataRecordId** – the ID of the Data Record entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of properties for Data Record
	Content Type:	application/json
Response:	**Add. Headers:**	–
	Content:	Result of request to update Data Record
	Content Type:	text/plain
	Status:	• 200 OK – Update of Data Record properties successfully requested (response of *"true"* for

		success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Data Record ID mismatch in JSON

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 JSON Record Content Lists of all the values for each Data Record.

Type:	*GET*		
URI:	/rest/serverengine/entity/datarecords/values		
Parameters:	Query: • **id** – the ID of each Data Record entity in Server • **recursive** – recurse all Data Tables within each Data Record and retrieve the values of any nested Data Records also (Default Value: false)		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Record Content Lists of the values for each Data Record	
	Content Type:	application/json	

Status:	• 200 OK – Data Record entity values successfully retrieved
	• 401 Unauthorized – Server authentication required
	• 403 Forbidden – Server authentication has failed or expired
	• 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 containing either:

- JSON Record Content Lists of all the values for each Data Record, or
- JSON Record Content Lists (Explicit Types) of all the values and data types for each Data Record

Type:	*POST*	
URI:	/rest/serverengine/entity/datarecords/values	
Parameters:	–	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Data Record Identifier List (with Parameters) specifying the Data Record entity IDs
	Content Type:	application/json
Response:	Add. Headers:	–
	Content:	JSON Record Content Lists of the values for each Data Record
	Content	application/json

Type:	
Status:	• 200 OK – Data Record entity values successfully retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 500 Internal Server Error – Server error or invalid Data Record ID specified

Response (Explicit Types):	**Add. Headers:**	–
	Content:	JSON Record Content Lists (Explicit Types) of the values and data types for each Data Record
	Content Type:	application/json
	Status:	• 200 OK – Data Record entity values and data types successfully retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 500 Internal Server Error – Server error or invalid Data Record ID specified

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.

Type:	PUT		
URI:	/rest/serverengine/entity/datarecords		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Record Content Lists (Fields Only) of the values for the Data Records	
	Content Type:	application/json	
Response:	Add. Headers:	–	
	Content:	–	
	Content Type:	–	
	Status:	• 200 OK – Values of Data Record entities successfully updated • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*PUT*		
URI:	/rest/serverengine/entity/datarecords/properties		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Name/Value Lists of the properties of the Data Records	
	Content Type:	application/json	
Response:	**Add. Headers:**	–	
	Content:	–	
	Content Type:	–	
	Status:	• 200 OK – Properties of Data Record entities successfully updated • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Data Record Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/datarecords/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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
Get All Data Sets	/entity/datasets	GET
Get Data Records for Data Set	/entity/datasets/{dataSetId}	GET
Delete Data Set Entity	/entity/datasets/{dataSetId}/delete	POST
Get Data Set Properties	/entity/datasets/{dataSetId}/properties	GET
Update Data Set Properties	/entity/datasets/{dataSetId}/properties	PUT
Service Version	/entity/datasets/version	GET

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.

Type:	*GET*		
URI:	/rest/serverengine/entity/datasets		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Identifier List of all the Data Sets in Server	
	Content Type:	application/json	
	Status:	• 200 OK – Identifier List of Data Sets returned • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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.

Type:	GET
URI:	/rest/serverengine/entity/datasets/{dataSetId}
Parameters:	Path: • **dataSetId** – the ID of the Data Set entity in Server

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	Add. Headers:	–
	Content:	JSON Identifier List of all the Data Records in Data Set
	Content Type:	application/json
	Status:	• 200 OK – Identifier List of Data Records returned • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*POST*		
URI:	/rest/serverengine/entity/datasets/{dataSetId}/delete		
Parameters:	Path: • **dataSetId** – the ID of the Data Set entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Result of request for Data Set removal	
	Content Type:	text/plain	
	Status:	• 200 OK – Deletion of Data Set successfully requested from Server (response of *"true"* for success or *"false"* for failure) • 401 Unauthorized – Server authentication	

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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.

Type:	*GET*		
URI:	/rest/serverengine/entity/datasets/{dataSetId}/properties		
Parameters:	Path: • **dataSetId** – the ID of the Data Set entity in Server		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	JSON Name/Value List (Properties Only) of properties for Data Set	
	Content Type:	application/json	
	Status:	• 200 OK – Data Set entity properties successfully retrieved • 401 Unauthorized – Server authentication	

		required
		• 403 Forbidden – Server authentication has failed or expired

PlanetPress.
Connect

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"*).

Type:	*PUT*		
URI:	/rest/serverengine/entity/datasets/{dataSetId}/properties		
Parameters:	Path: • **dataSetId** – the ID of the Data Set entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Name/Value List of properties for Data Set	
	Content Type:	application/json	
Response:	**Add. Headers:**	–	
	Content:	Result of request to update Data Set	
	Content Type:	text/plain	
	Status:	• 200 OK – Update of Data Set properties successfully requested (response of *"true"* for	

		success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Data Set ID mismatch in JSON

Service Version

Returns the version of the Data Set Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/datasets/version		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	200 OK – Version of REST Service retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired	

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	/workflow/datamining	GET
Process Data Mapping	/workflow/datamining/{configId}/{dataFileId}	POST
Process Data Mapping (JSON)	/workflow/datamining/{configId}	POST
Process Data Mapping (PDF/VT to Data Set)	/workflow/datamining/pdfvtds/{dataFileId}	POST
Process Data Mapping (PDF/VT to Content Set)	/workflow/datamining/pdfvtcs/{dataFileId}	POST
Get All Operations	/workflow/datamining/getOperations	GET
Get Progress of Operation	/workflow/datamining/getProgress/{operationId}	GET
Get Result of Operation	/workflow/datamining/getResult/{operationId}	POST
Cancel an Operation	/workflow/datamining/cancel/{operationId}	POST
Service Version	/workflow/datamining/version	GET

Service Handshake

Queries the availability of the Data Mapping service.

Type:	*GET*	
URI:	/rest/serverengine/workflow/datamining	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: DataMiningRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

Process Data Mapping

Submits a request to initiate a new Data Mapping 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.

Type:	*POST*
URI:	/rest/serverengine/workflow/datamining/{configId}/{dataFileId}
Parameters:	Path: • **configId** – the Managed File ID (or Name) of the Data Mapping configuration in File Store • **dataFileId** – the Managed File ID (or Name) of the data file in File Store Query: • **validate** – Only validate the Data Mapping operation to check for mapping errors (Default Value: false)
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be</td></tr></table>

			used to retrieve further information/cancel the operation
	Content:	–	
	Content Type:	–	
	Status:	• 202 Accepted – Creation of new operation successful • 400 Bad Request – Data file or Data Mapping Configuration not found in File Store • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	
Response (Validate):	**Add. Headers:**	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	
	Content:	–	
	Content Type:	–	
	Status:	• 202 Accepted – Creation of new operation successful • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

		• 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 (Managed File) of the data file's Managed File ID or Name 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.

Type:	*POST*
URI:	/rest/serverengine/workflow/datamining/{configId}
Parameters:	Path: • **configId** – the Managed File ID (or Name) of the Data Mapping configuration in File Store Query: • **validate** – Only validate the Data Mapping operation to check for mapping errors (Default Value: false)

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Identifier (Managed File) specifying Managed File ID or Name of Data file in File Store
	Content Type:	application/json

Response:	Add. Headers:	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be

			used to retrieve further information/cancel the operation
	Content:	–	
	Content Type:	–	
	Status:	202 Accepted – Creation of new operation successful 400 Bad Request – Data file or Data Mapping Configuration not found in File Store 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 500 Internal Server Error – JSON Identifier bad or missing	
Response (Validate):	**Add. Headers:**	**operationId** – Operation ID of new Data Mapping operation **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	
	Content:	–	
	Content Type:	–	
	Status:	202 Accepted – Creation of new operation successful 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired • 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 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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/datamining/pdfvtds/{dataFileId}	
Parameters:	Path: • **dataFileId** – the Managed File ID (or Name) of the PDF/VT data file in File Store	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	• **operationId** – Operation ID of new Data Mapping operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation

Content:	–
Content Type:	–
Status:	• 202 Accepted – Creation of new operation successful • 400 Bad Request – PDF/VT data file not found in File Store • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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 design 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.

Type:	*POST*
URI:	/rest/serverengine/workflow/datamining/pdfvtcs/{dataFileId}
Parameters:	Path: **dataFileId** – the Managed File ID (or Name) of the PDF/VT data file in File Store
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td> **operationId** – Operation ID of new Data Mapping operation **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation </td></tr></table>

Content:	–
Content Type:	–
Status:	• 202 Accepted – Creation of new operation successful • 400 Bad Request – PDF/VT data file not found in File Store • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*	
URI:	/rest/serverengine/workflow/datamining/getOperations	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	JSON Operations List of all the actively running operations in Server
	Content Type:	application/json
	Status:	• 200 OK – List of actively running operations successfully retrieved • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

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).

Type:	GET
URI:	/rest/serverengine/workflow/datamining/getProgress/{operationId}
Parameters:	Path: • **operationId** – Operation ID of Data Mapping operation

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	Add. Headers:	–
	Content:	Progress value of Data Mapping operation
	Content Type:	text/plain
	Status:	• 200 OK – Progress of operation successfully retrieved • 401 Unauthorized – Server authentication

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/datamining/getResult/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Data Mapping operation		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	Data Set ID (or Content Set ID)	
	Content Type:	text/plain	
	Status:	• 200 OK – Result of completed operation	

			successfully retrieved
			• 401 Unauthorized – Server authentication required
			• 403 Forbidden – Server authentication has failed or expired
Response (Validate):	**Add. Headers:**	–	
	Content:	JSON Data Mapping Validation Result	
	Content Type:	text/plain	
	Status:	• 200 OK – Result of completed operation successfully retrieved	
		• 401 Unauthorized – Server authentication required	
		• 403 Forbidden – Server authentication has failed or expired	

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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/datamining/cancel/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Data Mapping operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	–	
	Content Type:	–	
	Status:	• 204 No Content – Operation cancellation requested • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Data Mapping service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/datamining/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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	/entity/documents	GET
Get Document Metadata Properties	/entity/documents/ {documentId}/metadata	GET
Update Document Metadata Properties	/entity/documents/ {documentId}/metadata	PUT
Service Version	/entity/documents/version	GET

Service Handshake

Queries the availability of the Document Entity service.

Type:	*GET*	
URI:	/rest/serverengine/entity/documents	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: DocumentEntityRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*		
URI:	/rest/serverengine/entity/documents/{documentId}/metadata		
Parameters:	Path: • **documentId** – the ID of the Document entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Name/Value List (Properties Only) of metadata properties for Document	
	Content Type:	application/json	
	Status:	• 200 OK – Document entity metadata properties successfully retrieved • 401 Unauthorized – Server authentication	

		required
		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*PUT*
URI:	/rest/serverengine/entity/documents/{documentId}/metadata
Parameters:	Path: • **documentId** – the ID of the Document entity in Server
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>JSON Name/Value List of metadata properties for Document</td></tr><tr><td>**Content Type:**</td><td>application/json</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Result of request to update Document</td></tr><tr><td>**Content Type:**</td><td>text/plain</td></tr><tr><td>**Status:**</td><td>• 200 OK – Update of Document metadata</td></tr></table>

		properties successfully requested (response of *"true"* for success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Document ID mismatch in JSON

Service Version

Returns the version of the Document Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/documents/version		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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	/entity/documentsets	GET
Get Documents for Document Set	/entity/documentsets/{documentSetId}	GET
Get Document Set Metadata Properties	/entity/documentsets/ {documentSetId}/metadata	GET
Update Document Set Metadata Properties	/entity/documentsets/ {documentSetId}/metadata	PUT
Service Version	/entity/documentsets/version	GET

Service Handshake

Queries the availability of the Document Set Entity service.

Type:	*GET*	
URI:	/rest/serverengine/entity/documentsets	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: DocumentSetEntityRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*		
URI:	/rest/serverengine/entity/documentsets/{documentSetId}		
Parameters:	Path: • **documentSetId** – the ID of the Document Set entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Identifier List of all the Documents in Document Set	
	Content Type:	application/json	
	Status:	• 200 OK – Identifier List of Documents returned • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*
URI:	/rest/serverengine/entity/documentsets/{documentSetId}/metadata
Parameters:	Path: • **documentSetId** – the ID of the Document Set entity in Server

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	**Add. Headers:**	–
	Content:	JSON Name/Value List (Properties Only) of metadata properties for Document Set
	Content Type:	application/json
	Status:	• 200 OK – Document Set entity metadata properties successfully retrieved • 401 Unauthorized – Server authentication

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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"*).

Type:	*PUT*	
URI:	/rest/serverengine/entity/documentsets/{documentSetId}/metadata	
Parameters:	Path: • **documentSetId** – the ID of the Document Set entity in Server	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of metadata properties for Document Set
	Content Type:	application/json
Response:	Add. Headers:	–
	Content:	Result of request to update Document Set
	Content Type:	text/plain
	Status:	• 200 OK – Update of Document Set metadata

		properties successfully requested (response of *"true"* for success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Document Set ID mismatch in JSON

Service Version

Returns the version of the Document Set Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/documentsets/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

PlanetPress.
Connect

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	/workflow/contentcreation/email	GET
Process Content Creation (By Data Record) (JSON)	/workflow/contentcreation/email/{templateId}	POST
Process Content Creation (By Data) (JSON)	/workflow/contentcreation/email/{templateId}	POST
Get All Operations	/workflow/contentcreation/email/getOperations	GET
Get Progress of Operation	/workflow/contentcreation/email/getProgress/{operationId}	GET
Get Result of Operation	/workflow/contentcreation/email/getResult/{operationId}	POST
Cancel an Operation	/workflow/contentcreation/email/cancel/{operationId}	POST
Service Version	/workflow/contentcreation/email/version	GET

Service Handshake

Queries the availability of the Content Creation (Email) service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/contentcreation/email		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Handshake message: *Server Engine REST Service available: EmailExportRestService*	
	Content Type:	text/plain	
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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.

Type:	*POST*
URI:	/rest/serverengine/workflow/contentcreation/email/{templateId}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store Query: • **section** – the Section of the Email context to export (Defaults to default section in design template)

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Identifier List (with Email Parameters) specifying a list of Data Record entity IDs and parameters to be used for content creation
	Content Type:	application/json

Response:	Add. Headers:	• **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
	Content:	–
	Content Type:	–
	Status:	• 202 Accepted – Creation of new operation successful • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 404 Not Found – Design template or Data Record entity not found in File Store/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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/contentcreation/email/{templateId}		
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store Query: • **section** – the Section of the Email context to export (Defaults to default section in design template)		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	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	
	Content Type:	application/json	
Response:	**Add. Headers:**	• **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
	Content:	–
	Content Type:	–
	Status:	• 202 Accepted – Creation of new operation successful • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 404 Not Found – Design template 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.

Type:	GET		
URI:	/rest/serverengine/workflow/contentcreation/email/getOperations		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Operations List of all the actively running operations in Server	
	Content Type:	application/json	
	Status:	• 200 OK – List of actively running operations successfully retrieved • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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).

Type:	*GET*		
URI:	/rest/serverengine/workflow/contentcreation/email/getProgress/ {operationId}		
Parameters:	Path: • **operationId** – Operation ID of Content Creation (Email) operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Progress value of Content Creation (Email) operation	
	Content Type:	text/plain	

Status:	- 200 OK – Progress of operation successfully retrieved
	- 401 Unauthorized – Server authentication required
	- 403 Forbidden – Server authentication has failed or expired

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 report on the number of emails that were successfully sent.

Type:	*POST*	
URI:	/rest/serverengine/workflow/contentcreation/email/getResult/ {operationId}	
Parameters:	Path: • **operationId** – Operation ID of Content Creation (Email) operation	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Result of Content Creation (Email) Operation (with successful email count) (e.g. *"3 of 3 emails sent"*)
	Content Type:	text/plain
	Status:	• 200 OK – Result of completed operation

		successfully retrieved
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/contentcreation/email/cancel/{operationId}	
Parameters:	Path: • **operationId** – Operation ID of Content Creation (Email) operation	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	–
	Content Type:	–
	Status:	• 204 No Content – Operation cancellation requested • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Content Creation (Email) service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/contentcreation/email/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	200 OK – Version of REST Service retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired	

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	/entity	GET
Find Data Entity	/entity/find	PUT
Service Version	/entity/version	GET

PlanetPress.
Connect

Service Handshake

Queries the availability of the Entity service.

Type:	*GET*	
URI:	/rest/serverengine/entity	
Parameters:	–	
Requs	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: EntityRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

Find Data Entity

Submits data entity search criteria to the PlanetPress 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.

Type:	*PUT*		
URI:	/rest/serverengine/entity/find		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Search Parameters containing search criteria/rules	
	Content Type:	application/json	
Response:	**Add. Headers:**	–	
	Content:	JSON Identifier Lists (with Sort Key) containing all the entities matching the search criteria	
	Content Type:	application/json	
	Status:	• 200 OK – Search request successfully executed	

		401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 500 Internal Server Error – Server error or Invalid JSON structure specified

Service Version

Returns the version of the Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	200 OK – Version of REST Service retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired	

PlanetPress.
Connect

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	/filestore	GET
Download Managed File or Directory	/filestore/file/{fileId}	GET
Delete Managed File or Directory	/filestore/delete/{fileId}	GET
Upload Data Mapping Configuration	/filestore/DataMiningConfig	POST
Upload Job Creation Preset	/filestore/JobCreationConfig	POST
Upload Data File	/filestore/DataFile	POST
Upload Design Template	/filestore/template	POST
Upload Output Creation Preset	/filestore/OutputCreationConfig	POST
Service Version	/filestore/version	GET

Service Handshake

Queries the availability of the File Store service.

Type:	*GET*
URI:	/rest/serverengine/filestore
Parameters:	–
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Handshake message: *Server Engine REST Service available: FilestoreRestService*</td></tr><tr><td>**Content Type:**</td><td>text/plain</td></tr><tr><td>**Status:**</td><td> 200 OK – REST Service available 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired </td></tr></table>

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).

Type:	*GET*
URI:	/rest/serverengine/filestore/file/{fileId}
Parameters:	Path: • **fileId** – the Managed File ID (or Name) of the file or directory in File Store
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>Content-Disposition • File – *"attachment; filename={OrigFileName}"* • Directory – *"attachment; filename={OrigDirName}.zip"*</td></tr><tr><td>**Content:**</td><td>File or Directory (zipped as file)</td></tr><tr><td>**Content Type:**</td><td>• File – application/octet-stream • Directory – application/zip</td></tr></table>

Status:	• 200 OK – File or directory successfully downloaded from File Store • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 404 Not Found – File or 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 (*"true"* or *"false"*).

Type:	*GET*	
URI:	/rest/serverengine/filestore/delete/{fileId}	
Parameters:	Path: • **fileId** – the Managed File ID (or Name) of the file or directory in File Store	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Result of request for removal
	Content Type:	text/plain
	Status:	• 200 OK – Removal of file or directory successfully requested from File Store

		(response of *"true"* for success or *"false"* for failure)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 404 Not Found – File or directory not found in File Store

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.

Type:	*POST*
URI:	/rest/serverengine/filestore/DataMiningConfig
Parameters:	Query: • **filename** – the file name of the configuration to be uploaded (No Default Value) • **persistent** – whether the configuration to be uploaded will be persistent in File Store (Default Value: false)
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>Data Mapping Configuration (File)</td></tr><tr><td>**Content Type:**</td><td>application/octet-stream</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Managed File ID</td></tr><tr><td>**Content Type:**</td><td>text/plain</td></tr><tr><td>**Status:**</td><td>• 200 OK – Configuration successfully uploaded</td></tr></table>

		to File Store
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*
URI:	/rest/serverengine/filestore/JobCreationConfig
Parameters:	Query: • **filename** – the file name of the preset to be uploaded (No Default Value) • **persistent** – whether the preset to be uploaded will be persistent in File Store (Default Value: false)
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>Job Creation Preset (File)</td></tr><tr><td>**Content Type:**</td><td>application/xml</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Managed File ID</td></tr><tr><td>**Content Type:**</td><td>text/plain</td></tr><tr><td>**Status:**</td><td>• 200 OK – Preset successfully uploaded to File</td></tr></table>

		Store
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*
URI:	/rest/serverengine/filestore/DataFile
Parameters:	Query: • **filename** – the file name of the data file to be uploaded (No Default Value) • **persistent** – whether the data file to be uploaded will be persistent in File Store (Default Value: false)
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>Data File (File)</td></tr><tr><td>**Content Type:**</td><td>application/octet-stream</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Managed File ID</td></tr><tr><td>**Content Type:**</td><td>text/plain</td></tr><tr><td>**Status:**</td><td>• 200 OK – Data file successfully uploaded to</td></tr></table>

	File Store
	• 401 Unauthorized – Server authentication required
	• 403 Forbidden – Server authentication has failed or expired

Upload Design Template

Submits a design 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 design template.

Type:	*POST*	
URI:	/rest/serverengine/filestore/template	
Parameters:	Query: • **filename** – the file name of the design template to be uploaded (No Default Value) • **persistent** – whether the design template to be uploaded will be persistent in File Store (Default Value: false)	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	Design Template (File)
	Content Type:	application/zip
Response:	**Add. Headers:**	–
	Content:	Managed File ID
	Content Type:	text/plain
	Status:	• 200 OK – Template successfully uploaded to

		File Store - 401 Unauthorized – Server authentication required - 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*
URI:	/rest/serverengine/filestore/OutputCreationConfig
Parameters:	Query: • **filename** – the file name of the preset to be uploaded (No Default Value) • **persistent** – whether the preset to be uploaded will be persistent in File Store (Default Value: false)

Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	Output Creation Preset (File)
	Content Type:	application/xml

Response:	Add. Headers:	–
	Content:	Managed File ID
	Content Type:	text/plain
	Status:	• 200 OK – Preset successfully uploaded to File

		Store
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the File Store service.

Type:	*GET*	
URI:	/rest/serverengine/filestore/version	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Version of Service
	Content Type:	text/plain
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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	/workflow/contentcreation/html	GET
Process Content Creation (By Data Record)	/workflow/contentcreation/html/ {templateId}/{dataRecordId: [0-9]+}	GET
Process Content Creation (By Data Record) (JSON)	/workflow/contentcreation/html/ {templateId}/{dataRecordId: [0-9]+}	POST
Process Content Creation (By Data) (JSON)	/workflow/contentcreation/html/ {templateId}	POST
Process Content Creation (No Data)	/workflow/contentcreation/html/ {templateId}	POST
Get Template Resource	/workflow/contentcreation/html/ {templateId}/{relPath: .+}	GET
Service Version	/workflow/contentcreation/html/version	GET

Service Handshake

Queries the availability of the Content Creation (HTML) service.

Type:	*GET*	
URI:	/rest/serverengine/workflow/contentcreation/html	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: HTMLMergeRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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 section specified.

Type:	*GET*
URI:	/rest/serverengine/workflow/contentcreation/html/{templateId}/ {dataRecordId: [0-9]+}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store • **dataRecordId** – the ID of the Data Record entity in Server Query: • **section** – the section within the Web context to create (Defaults to default section in design template) • **inline** – the inline mode to be used in the creation of content (Possible values: NONE, CSS or ALL. Default Value: NONE)

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	**Add. Headers:**	–

Content:	HTML output for the Data Record
Content Type:	text/html
Status:	200 OK – Output created successfully 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Design 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

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 List as content, and on success returns a response containing the HTML output produced, specific to the Data Record specified.

Type:	*POST*
URI:	/rest/serverengine/workflow/contentcreation/html/{templateId}/ {dataRecordId: [0-9]+}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store • **dataRecordId** – the ID of the Data Record entity in Server

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON HTML Parameters List specifying parameters to be used for content creation
	Content Type:	application/json

Response:	**Add. Headers:**	–
	Content:	HTML output for the Data Record
	Content Type:	text/html

Status:	200 OK – Output created successfully 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Design 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

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 specified.

Type:	*POST*
URI:	/rest/serverengine/workflow/contentcreation/html/{templateId}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store Query: • **section** – the section within the Web context to create (Defaults to default section in design template) • **inline** – the inline mode to be used in the creation of content (Possible values: NONE, CSS or ALL. Default Value: NONE)

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Record Data List specifying a list of data values for the Data Record
	Content Type:	application/json

Response:	**Add. Headers:**	–

Content:	HTML output for the Data Record
Content Type:	text/html
Status:	200 OK – Output created successfully 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Design template not found in File Store 500 Internal Server Error – Content Creation Error: Web context in template not found

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 design template and no input data.

Type:	*POST*
URI:	/rest/serverengine/workflow/contentcreation/html/{templateId}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store Query: • **section** – the section within the Web context to create (Defaults to default section in design template) • **inline** – the inline mode to be used in the creation of content (Possible values: NONE, CSS or ALL. Default Value: NONE)

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	**Add. Headers:**	–
	Content:	HTML output from only the template (no data)

Content Type:	text/html
Status:	200 OK – Output created successfully 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Design template not found in File Store 500 Internal Server Error – Content Creation Error: Web context in template not found

Get Template Resource

Submits a request to retrieve a resource from a design template stored in the File Store.

Request takes no content, and on success returns a response containing the resource from the design template.

Type:	*GET*
URI:	/rest/serverengine/workflow/contentcreation/html/{templateId}/{relPath:.+}
Parameters:	Path: • **templateId** – the Managed File ID (or Name) of the design template in File Store • **relPath** – the relative path to the resource within the design template

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	**Add. Headers:**	–
	Content:	Resource located at the relative path within template
	Content Type:	(Depends on Resource requested)

Status:	• 200 OK – Resource successfully retrieved • 400 Bad Request – Unable to open resource within template or resource doesn't exist • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 404 Not Found – Design template not found in File Store • 500 Internal Server Error – Unable to open template or template doesn't exist

Service Version

Returns the version of the Content Creation (HTML) service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/contentcreation/html/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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	/workflow/jobcreation	GET
Process Job Creation	/workflow/jobcreation/{configId}	POST
Process Job Creation (JSON)	/workflow/jobcreation/{configId}	POST
Process Job Creation (JSON Job Set Structure)	/workflow/jobcreation	POST
Get All Operations	/workflow/jobcreation/getOperations	GET
Get Progress of Operation	/workflow/jobcreation/getProgress/ {operationId}	GET
Get Result of Operation	/workflow/jobcreation/getResult/ {operationId}	POST
Cancel an Operation	/workflow/jobcreation/cancel/ {operationId}	POST
Service Version	/workflow/jobcreation/version	GET

Service Handshake

Queries the availability of the Job Creation service.

Type:	*GET*
URI:	/rest/serverengine/workflow/jobcreation
Parameters:	–

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: JobCreationRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/jobcreation/{configId}	
Parameters:	Path: • **configId** – the Managed File ID (or Name) of the Job Creation Preset in File Store	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	Add. Headers:	• **operationId** – Operation ID of new Job Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation
	Content:	–
	Content Type:	–

Status:	• 202 Accepted – Creation of new operation successful
	• 401 Unauthorized – Server authentication required
	• 403 Forbidden – Server authentication has failed or expired
	• 404 Not Found – Job Creation Preset not found in File Store

Process Job Creation (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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/jobcreation/{configId}	
Parameters:	Path: • **configId** – the Managed File ID (or Name) of the Job Creation Preset in File Store	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Identifier List specifying a list of Content Set entity IDs
	Content Type:	application/json
Response:	Add. Headers:	• **operationId** – Operation ID of new Job Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation
	Content:	–
	Content	–

Type:	
Status:	202 Accepted – Creation of new operation successful 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 404 Not Found – Job Creation Preset or Content Set entity not found in File Store/Server

Process Job Creation (JSON Job Set Structure)

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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/jobcreation	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Job Set Structure describing Job Set (and Content Items)
	Content Type:	application/json
Response:	**Add. Headers:**	• **operationId** – Operation ID of new Job Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation
	Content:	–
	Content Type:	–

	Status:	• 202 Accepted – Creation of new operation successful • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	GET		
URI:	/rest/serverengine/workflow/jobcreation/getOperations		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	JSON Operations List of all the actively running operations in Server	
	Content Type:	application/json	
	Status:	• 200 OK – List of actively running operations successfully retrieved • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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).

Type:	*GET*		
URI:	/rest/serverengine/workflow/jobcreation/getProgress/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Job Creation operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Progress value of Job Creation operation	
	Content Type:	text/plain	
	Status:	• 200 OK – Progress of operation successfully retrieved • 401 Unauthorized – Server authentication	

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/jobcreation/getResult/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Job Creation operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Job Set ID	
	Content Type:	text/plain	
	Status:	• 200 OK – Result of completed operation successfully retrieved • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/jobcreation/cancel/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Job Creation operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	–	
	Content Type:	–	
	Status:	• 204 No Content – Operation cancellation requested • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Job Creation service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/jobcreation/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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	/entity/jobs	GET
Get Content Items for Job	/entity/jobs/{jobId}/contents	GET
Get Job Segments for Job	/entity/jobs/{jobId}	GET
Get Job Metadata Properties	/entity/jobs/{jobId}/metadata	GET
Update Job Metadata Properties	/entity/jobs/{jobId}/metadata	PUT
Get Job Properties	/entity/jobs/{jobId}/properties	GET
Update Job Properties	/entity/jobs/{jobId}/properties	PUT
Update Multiple Job Properties	/entity/jobs/properties	PUT
Service Version	/entity/jobs/version	GET

Service Handshake

Queries the availability of the Job Entity service.

Type:	*GET*
URI:	/rest/serverengine/entity/jobs
Parameters:	–
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Handshake message: *Server Engine REST Service available: JobEntityRestService*</td></tr><tr><td>**Content Type:**</td><td>text/plain</td></tr><tr><td>**Status:**</td><td> 200 OK – REST Service available 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired </td></tr></table>

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.

Type:	*GET*	
URI:	/rest/serverengine/entity/jobs/{jobId}/contents	
Parameters:	Path: • **jobId** – the ID of the Job entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	JSON Content Item Identifier List of all the Content Items in Job
	Content Type:	application/json
	Status:	• 200 OK – Content Item Identifier List returned • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	GET		
URI:	/rest/serverengine/entity/jobs/{jobId}		
Parameters:	Path: • **jobId** – the ID of the Job entity in Server		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	JSON Identifier List of all the Job Segments in Job	
	Content Type:	application/json	
	Status:	• 200 OK – Identifier List of Job Segments returned • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*	
URI:	/rest/serverengine/entity/jobs/{jobId}/metadata	
Parameters:	Path: • **jobId** – the ID of the Job entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	JSON Name/Value List (Properties Only) of metadata properties for Job
	Content Type:	application/json
	Status:	• 200 OK – Job entity metadata properties successfully retrieved • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*PUT*	
URI:	/rest/serverengine/entity/jobs/{jobId}/metadata	
Parameters:	Path: • **jobId** – the ID of the Job entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of metadata properties for Job
	Content Type:	application/json
Response:	**Add. Headers:**	–
	Content:	Result of request to update Job
	Content Type:	text/plain
	Status:	• 200 OK – Update of Job metadata properties

	successfully requested (response of *"true"* for success)
	• 401 Unauthorized – Server authentication required
	• 403 Forbidden – Server authentication has failed or expired
	• 500 Internal Server Error – Server error or Job ID mismatch in JSON

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.

Type:	*GET*	
URI:	/rest/serverengine/entity/jobs/{jobId}/properties	
Parameters:	Path: • **jobId** – the ID of the Job entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	JSON Name/Value List (Properties Only) of properties for Job
	Content Type:	application/json
	Status:	• 200 OK – Job entity properties successfully retrieved • 401 Unauthorized – Server authentication

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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"*).

Type:	*PUT*	
URI:	/rest/serverengine/entity/jobs/{jobId}/properties	
Parameters:	Path: • **jobId** – the ID of the Job entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of properties for Job
	Content Type:	application/json
Response:	**Add. Headers:**	–
	Content:	Result of request to update Job
	Content Type:	text/plain
	Status:	• 200 OK – Update of Job properties successfully requested (response of *"true"* for success) • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired
		• 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.

Type:	*PUT*
URI:	/rest/serverengine/entity/jobs/properties
Parameters:	–

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value Lists of the properties of the Jobs
	Content Type:	application/json

Response:	**Add. Headers:**	–
	Content:	–
	Content Type:	–
	Status:	• 200 OK – Properties of Job entities successfully updated • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Job Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/jobs/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved	
		• 401 Unauthorized – Server authentication required	
		• 403 Forbidden – Server authentication has failed or expired	

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	/entity/jobsegments	GET
Get Document Sets for Job Segment	/entity/jobsegments/{jobSegmentId}	GET
Get Job Segment Metadata Properties	/entity/jobsegments/{jobSegmentId}/metadata	GET
Update Job Segment Metadata Properties	/entity/jobsegments/{jobSegmentId}/metadata	PUT
Service Version	/entity/jobsegments/version	GET

Service Handshake

Queries the availability of the Job Segment Entity service.

Type:	*GET*
URI:	/rest/serverengine/entity/jobsegments
Parameters:	–

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: JobSegmentEntityRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*	
URI:	/rest/serverengine/entity/jobsegments/{jobSegmentId}	
Parameters:	Path: • **jobSegmentId** – the ID of the Job Segment entity in Server	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	Add. Headers:	–
	Content:	JSON Identifier List of all the Document Sets in Job Segment
	Content Type:	application/json
	Status:	• 200 OK – Identifier List of Document Sets returned • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*GET*	
URI:	/rest/serverengine/entity/jobsegments/{jobSegmentId}/metadata	
Parameters:	Path: • **jobSegmentId** – the ID of the Job Segment entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	JSON Name/Value List (Properties Only) of metadata properties for Job Segment
	Content Type:	application/json
	Status:	• 200 OK – Job Segment entity metadata properties successfully retrieved • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*PUT*
URI:	/rest/serverengine/entity/jobsegments/{jobSegmentId}/metadata
Parameters:	Path: • **jobSegmentId** – the ID of the Job Segment entity in Server

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of metadata properties for Job Segment
	Content Type:	application/json

Response:	**Add. Headers:**	–
	Content:	Result of request to update Job Segment
	Content Type:	text/plain
	Status:	• 200 OK – Update of Job Segment metadata

		properties successfully requested (response of *"true"* for success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Job Segment ID mismatch in JSON

Service Version

Returns the version of the Job Segment Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/jobsegments/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	200 OK – Version of REST Service retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired	

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
Get All Job Sets	/entity/jobsets	GET
Get Jobs for Job Set	/entity/jobsets/{jobSetId}	GET
Delete Job Set Entity	/entity/jobsets/{jobSetId}/delete	POST
Get Job Set Metadata Properties	/entity/jobsets/{jobSetId}/metadata	GET
Update Job Set Metadata Properties	/entity/jobsets/{jobSetId}/metadata	PUT
Get Job Set Properties	/entity/jobsets/{jobSetId}/properties	GET
Update Job Set Properties	/entity/jobsets/{jobSetId}/properties	PUT
Service Version	/entity/jobsets/version	GET

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.

Type:	GET		
URI:	/rest/serverengine/entity/jobsets		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	JSON Identifier List of all the Job Sets in Server	
	Content Type:	application/json	
	Status:	• 200 OK – Identifier List of Job Sets returned • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

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.

Type:	*GET*	
URI:	/rest/serverengine/entity/jobsets/{jobSetId}	
Parameters:	Path: • **jobSetId** – the ID of the Job Set entity in Server	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	JSON Identifier List of all the Jobs in Job Set
	Content Type:	application/json
	Status:	• 200 OK – Identifier List of Jobs returned • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*POST*		
URI:	/rest/serverengine/entity/jobsets/{jobSetId}/delete		
Parameters:	Path: • **jobSetId** – the ID of the Job Set entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Result of request for Job Set removal	
	Content Type:	text/plain	
	Status:	• 200 OK – Deletion of Job Set successfully requested from Server (response of *"true"* for success or *"false"* for failure) • 401 Unauthorized – Server authentication	

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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.

Type:	*GET*
URI:	/rest/serverengine/entity/jobsets/{jobSetId}/metadata
Parameters:	Path: • **jobSetId** – the ID of the Job Set entity in Server

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–

Response:	**Add. Headers:**	–
	Content:	JSON Name/Value List (Properties Only) of metadata properties for Job Set
	Content Type:	application/json
	Status:	• 200 OK – Job Set entity metadata properties successfully retrieved • 401 Unauthorized – Server authentication

		required
		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*PUT*
URI:	/rest/serverengine/entity/jobsets/{jobSetId}/metadata
Parameters:	Path: • **jobSetId** – the ID of the Job Set entity in Server

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Name/Value List of metadata properties for Job Set
	Content Type:	application/json

Response:	**Add. Headers:**	–
	Content:	Result of request to update Job Set
	Content Type:	text/plain
	Status:	• 200 OK – Update of Job Set metadata

		properties successfully requested (response of *"true"* for success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Job Set ID mismatch in JSON

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.

Type:	*GET*
URI:	/rest/serverengine/entity/jobsets/{jobSetId}/properties
Parameters:	Path: • **jobSetId** – the ID of the Job Set entity in Server
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>JSON Name/Value List (Properties Only) of properties for Job Set</td></tr><tr><td>**Content Type:**</td><td>application/json</td></tr><tr><td>**Status:**</td><td>• 200 OK – Job Set entity properties successfully retrieved • 401 Unauthorized – Server authentication</td></tr></table>

		required
		• 403 Forbidden – Server authentication has failed or expired

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"*).

Type:	*PUT*		
URI:	/rest/serverengine/entity/jobsets/{jobSetId}/properties		
Parameters:	Path: • **jobSetId** – the ID of the Job Set entity in Server		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Name/Value List of properties for Job Set	
	Content Type:	application/json	
Response:	**Add. Headers:**	–	
	Content:	Result of request to update Job Set	
	Content Type:	text/plain	
	Status:	• 200 OK – Update of Job Set properties successfully requested (response of *"true"* for	

		success)
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 500 Internal Server Error – Server error or Job Set ID mismatch in JSON

Service Version

Returns the version of the Job Set Entity service.

Type:	*GET*		
URI:	/rest/serverengine/entity/jobsets/version		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

PlanetPress.
Connect

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	/workflow/outputcreation	GET
Process Output Creation	/workflow/outputcreation/{configId}/ {jobSetId}	POST
Process Output Creation (JSON)	/workflow/outputcreation/{configId}	POST
Process Output Creation (By Job) (JSON)	/workflow/outputcreation/{configId}/jobs	POST
Run +PReS Enhance Workflow Configuration	/workflow/outputcreation/execute/{args}/ {size}	GET
Get All Operations	/workflow/outputcreation/getOperations	GET
Get Progress of Operation	/workflow/outputcreation/getProgress/ {operationId}	GET
Get Result of Operation	/workflow/outputcreation/getResult/ {operationId}	POST
Get Result of Operation (as Text)	/workflow/outputcreation/getResultTxt/ {operationId}	POST
Cancel an Operation	/workflow/outputcreation/cancel/ {operationId}	POST
Service Version	/workflow/outputcreation/version	GET

Service Handshake

Queries the availability of the Output Creation service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/outputcreation		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	Add. Headers:	–	
	Content:	Handshake message: *Server Engine REST Service available: OutputCreationRestService*	
	Content Type:	text/plain	
	Status:	• 200 OK – REST Service available	
		• 401 Unauthorized – Server authentication required	
		• 403 Forbidden – Server authentication has failed or expired	

Process Output Creation

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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/outputcreation/{configId}/{jobSetId}	
Parameters:	Path: • **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	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	• **operationId** – Operation ID of new Output Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation
	Content:	–
	Content	–

Type:	
Status:	• 202 Accepted – Creation of new operation successful
	• 401 Unauthorized – Server authentication required
	• 403 Forbidden – Server authentication has failed or expired
	• 404 Not Found – Output Creation Preset or Job Set entity not found in File Store/Server

Process Output Creation (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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/outputcreation/{configId}		
Parameters:	Path: • **configId** – the Managed File ID (or Name) of the Output Creation Preset in File Store		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON Identifier (with Output Parameters) specifying the Job Set entity's ID	
	Content Type:	application/json	
Response:	**Add. Headers:**	• **operationId** – Operation ID of new Output Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation	
	Content:	–	

Content Type:	–	
Status:	202 Accepted – Creation of new operation successful 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 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.

Type:	*POST*
URI:	/rest/serverengine/workflow/outputcreation/{configId}/jobs
Parameters:	Path: • **configId** – the Managed File ID (or Name) of the Output Creation Preset in File Store

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	JSON Identifier List (with Output Parameters) specifying the Job entity IDs
	Content Type:	application/json

Response:	**Add. Headers:**	• **operationId** – Operation ID of new Output Creation operation • **Link** – Contains multiple link URLs that can be used to retrieve further information/cancel the operation
	Content:	–

	Content Type:	–
	Status:	202 Accepted – Creation of new operation successful 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired 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.

Type:	*GET*		
URI:	/rest/serverengine/workflow/outputcreation/execute/{args}/{size}		

Parameters:	Path:
	
	• **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 PlanetPress Connect to determine the job size (which determines the number of speed units to use)		

Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	

---

[1]For a list of all the available command-line arguments accepted by the Weaver engine, please reference the +PReS Enhance documentation.

Response:	Add. Headers:	–
	Content:	Result of request to run +PReS Enhance workflow
	Content Type:	text/plain
	Status:	200 OK – +PReS Enhance Workflow Configuration successfully run/executed 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired

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.

Type:	GET	
URI:	/rest/serverengine/workflow/outputcreation/getOperations	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	JSON Operations List of all the actively running operations in Server
	Content Type:	application/json
	Status:	• 200 OK – List of actively running operations successfully retrieved • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

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).

Type:	*GET*	
URI:	/rest/serverengine/workflow/outputcreation/getProgress/{operationId}	
Parameters:	Path: • **operationId** – Operation ID of Output Creation operation	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Progress value of Output Creation operation
	Content Type:	text/plain
	Status:	• 200 OK – Progress of operation successfully retrieved • 401 Unauthorized – Server authentication

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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).

Type:	*POST*		
URI:	/rest/serverengine/workflow/outputcreation/getResult/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Output Creation operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Absolute Paths of the Output Files or the Output File itself	
	Content Type:	application/octet-stream	
	Status:	• 200 OK – Result of completed operation successfully retrieved	

		- 401 Unauthorized – Server authentication required - 403 Forbidden – Server authentication has failed or expired

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).

Type:	*POST*		
URI:	/rest/serverengine/workflow/outputcreation/getResultTxt/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of Output Creation operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Absolute Path(s) of the Output File(s)	
	Content Type:	text/plain	
	Status:	• 200 OK – Result of completed operation successfully retrieved • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*	
URI:	/rest/serverengine/workflow/outputcreation/cancel/{operationId}	
Parameters:	Path: • **operationId** – Operation ID of Output Creation operation	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	–
	Content Type:	–
	Status:	• 204 No Content – Operation cancellation requested • 401 Unauthorized – Server authentication required

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the Output Creation service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/outputcreation/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	200 OK – Version of REST Service retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired	

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	/workflow/print	GET
Process All-In-One (JSON)	/workflow/print/submit	POST
Process All-In-One (Adhoc Data)	/workflow/print/{dmConfigId}/{templateId}/{jcConfigId}/{ocConfigId}	POST
Get All Operations	/workflow/print/getOperations	GET
Get Progress of Operation	/workflow/print/getProgress/{operationId}	GET
Get Result of Operation	/workflow/print/getResult/{operationId}	POST
Get Result of Operation (as Text)	/workflow/print/getResultTxt/{operationId}	POST
Cancel an Operation	/workflow/print/cancel/{operationId}	POST
Service Version	/workflow/print/version	GET

Service Handshake

Queries the availability of the All-In-One service.

Type:	*GET*	
URI:	/rest/serverengine/workflow/print	
Parameters:	–	
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	**Add. Headers:**	–
	Content:	Handshake message: *Server Engine REST Service available: PrintRestService*
	Content Type:	text/plain
	Status:	• 200 OK – REST Service available • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/print/submit		
Parameters:	–		
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	JSON All-In-One Configuration containing workflow processes/parameters	
	Content Type:	application/json	
Response:	Add. Headers:	• **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	
	Content:	–	
	Content Type:	–	
	Status:	• 202 Accepted – Creation of new operation	

		successful
		• 400 Bad Request – Required Input resource/file not found in File Store
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 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)

Type:	*POST*
URI:	/rest/serverengine/workflow/print/{dmConfigId}/{templateId}/{jcConfigId}/{ocConfigId}
Parameters:	Path: • **dmConfigId** – the Managed File ID (or Name) of the Data Mapping configuration in File Store • **templateId** – the Managed File ID (or Name) of the design template in File Store • **jcConfigId** – the Managed File ID (or Name) of the Job Creation Preset in File Store (Optional – the value of "0" can be specified if no preset is to be used) • **ocConfigId** – the Managed File ID (or Name) of the Output Creation Preset in File Store Query: • **async** – whether to run the operation asynchronously (Default

<table>
<tr>
<td></td>
<td colspan="2">

Value: true)

- **resultAsTxt** – whether to retrieve the result as text (Synchronous Only) (Default Value: false)
- **createOnly** – whether output is to be only created in the server and not sent to it's final destination (Default Value: false)
- **printRange** – a specific range of records in the input data file to restrict the print output to (No Default Value)
- **filename** – the file name of the data file to be uploaded (No Default Value)

</td>
</tr>
<tr>
<td>**Request:**</td>
<td>**Add. Headers:**</td>
<td>**auth_token** – Authorization Token (if server security settings enabled)</td>
</tr>
<tr>
<td></td>
<td>**Content:**</td>
<td>Data File (File)</td>
</tr>
<tr>
<td></td>
<td>**Content Type:**</td>
<td>application/octet-stream</td>
</tr>
<tr>
<td>**Response:**</td>
<td>**Add. Headers:**</td>
<td>

- **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

</td>
</tr>
<tr>
<td></td>
<td>**Content:**</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>**Content Type:**</td>
<td>–</td>
</tr>
<tr>
<td></td>
<td>**Status:**</td>
<td>

- 202 Accepted – Data file successfully uploaded to File Store and creation of new operation successful
- 400 Bad Request – Unable to locate one or more inputs in File Store with Managed

</td>
</tr>
</table>

		File ID(s) and/or Name(s) specified
		• 401 Unauthorized – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 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)
Response (Synchronous):	**Add. Headers:**	–
	Content:	Absolute Paths of the Output Files or the Output File itself
	Content Type:	application/octet-stream
	Status:	• 200 OK – Data file uploaded to File Store, and a new operation was created and completed successfully with the result returned
		• 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 – Server authentication required
		• 403 Forbidden – Server authentication has failed or expired
		• 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)
Response (Synchronous + Get Result as Text):	**Add. Headers:**	–
	Content:	Absolute Path(s) of the Output File(s)
	Content Type:	text/plain
	Status:	• 200 OK – Data file uploaded to File Store, and a new operation was created and completed successfully with the result returned • 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 – Server authentication required • 403 Forbidden – Server authentication has failed or expired • 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)

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.

Type:	*GET*		
URI:	/rest/serverengine/workflow/print/getOperations		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	JSON Operations List of all the actively running operations in Server	
	Content Type:	application/json	
	Status:	• 200 OK – List of actively running operations successfully retrieved	
		• 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

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).

Type:	*GET*		
URI:	/rest/serverengine/workflow/print/getProgress/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of All-In-One operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Progress value of All-In-One operation	
	Content Type:	text/plain	
	Status:	• 200 OK – Progress of operation successfully retrieved • 401 Unauthorized – Server authentication	

| | | required |
| | | • 403 Forbidden – Server authentication has failed or expired |

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).

Type:	*POST*
URI:	/rest/serverengine/workflow/print/getResult/{operationId}
Parameters:	Path: • **operationId** – Operation ID of All-In-One operation
Request:	<table><tr><td>**Add. Headers:**</td><td>**auth_token** – Authorization Token (if server security settings enabled)</td></tr><tr><td>**Content:**</td><td>–</td></tr><tr><td>**Content Type:**</td><td>–</td></tr></table>
Response:	<table><tr><td>**Add. Headers:**</td><td>–</td></tr><tr><td>**Content:**</td><td>Either: • the ID of the Data Set, Content Set or Job Set, or • the Absolute Paths of the Output Files or the</td></tr></table>

	Output File itself
Content Type:	application/octet-stream
Status:	• 200 OK – Result of completed operation successfully retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired

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).

Type:	*POST*	
URI:	/rest/serverengine/workflow/print/getResultTxt/{operationId}	
Parameters:	Path: • **operationId** – Operation ID of All-In-One operation	
Request:	Add. Headers:	**auth_token** – Authorization Token (if server security settings enabled)
	Content:	–
	Content Type:	–
Response:	Add. Headers:	–
	Content:	Either: • the ID of the Data Set, Content Set or Job Set, or • the Absolute Path(s) of the Output File(s)

	Content Type:	text/plain
	Status:	200 OK – Result of completed operation successfully retrieved 401 Unauthorized – Server authentication required 403 Forbidden – Server authentication has failed or expired

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.

Type:	*POST*		
URI:	/rest/serverengine/workflow/print/cancel/{operationId}		
Parameters:	Path: • **operationId** – Operation ID of All-In-One operation		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	–	
	Content Type:	–	
	Status:	• 204 No Content – Operation cancellation requested • 401 Unauthorized – Server authentication required	

		• 403 Forbidden – Server authentication has failed or expired

Service Version

Returns the version of the All-In-One service.

Type:	*GET*		
URI:	/rest/serverengine/workflow/print/version		
Parameters:	–		
Request:	**Add. Headers:**	**auth_token** – Authorization Token (if server security settings enabled)	
	Content:	–	
	Content Type:	–	
Response:	**Add. Headers:**	–	
	Content:	Version of Service	
	Content Type:	text/plain	
	Status:	• 200 OK – Version of REST Service retrieved • 401 Unauthorized – Server authentication required • 403 Forbidden – Server authentication has failed or expired	

Copyright Information

PlanetPress.
Connect

Legal Notices and Acknowledgements

PlanetPress Connect, Copyright © 2021, Objectif Lune Inc. All rights reserved.

This guide uses the following third party components:

- **jQuery Library** Copyright © JS Foundation and other contributors. This is distributed under the terms of the Massachusetts Institute of Technology (MIT) license.
- **QUnit Library** Copyright © JS/jQuery Foundation and other contributors. This is distributed under the terms of the Massachusetts Institute of Technology (MIT) license.