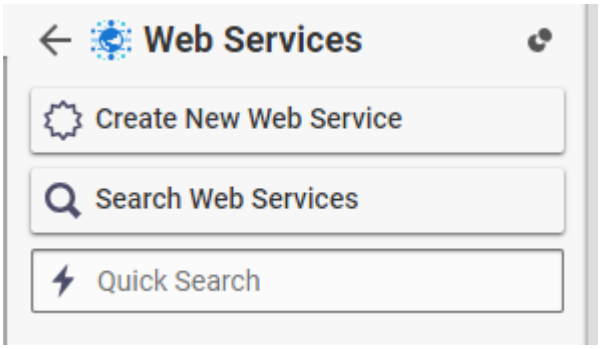
Web Service

Creating a New Web Service

The following steps outline the process of configuring a new Web Service:

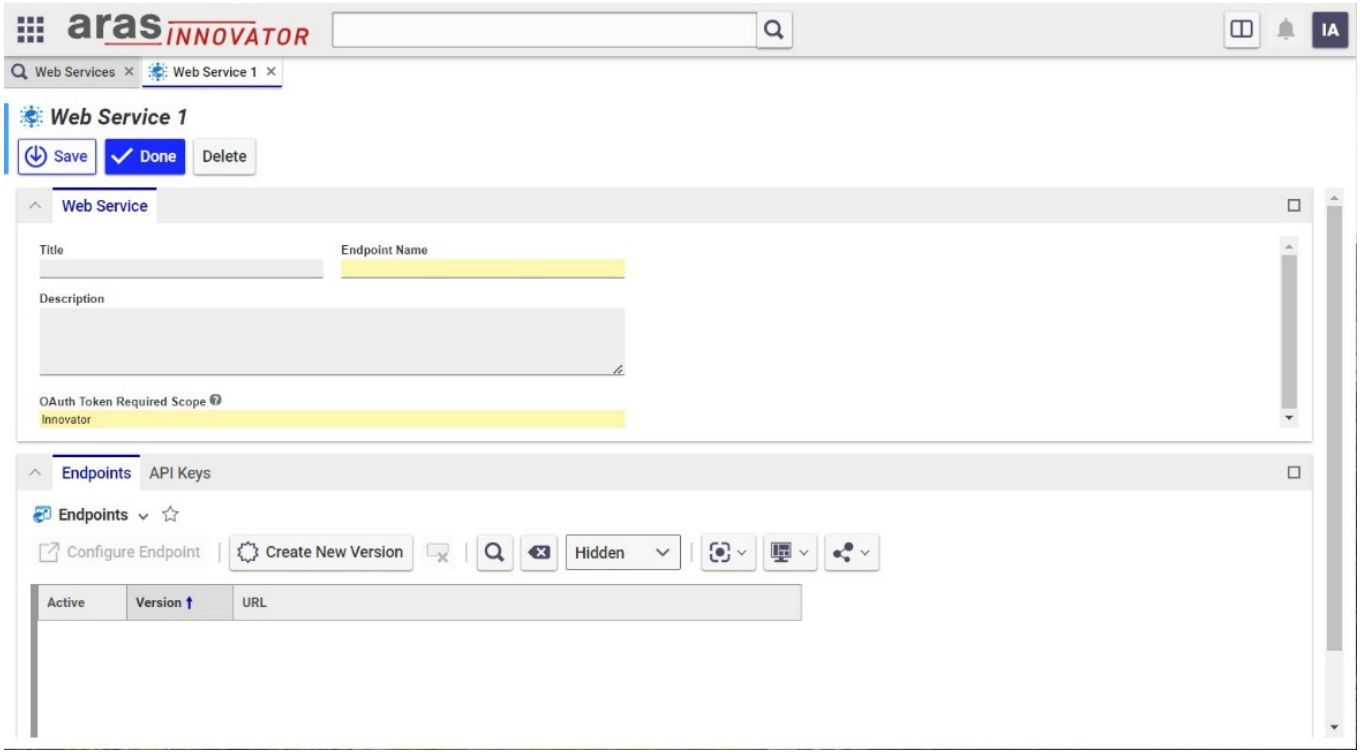
1. From the **Table of Contents**, expand **Administrator** and select **External Access**.
2. Expand **External Access** and select **Web Services**.

   The **Web Services TOC** appears.

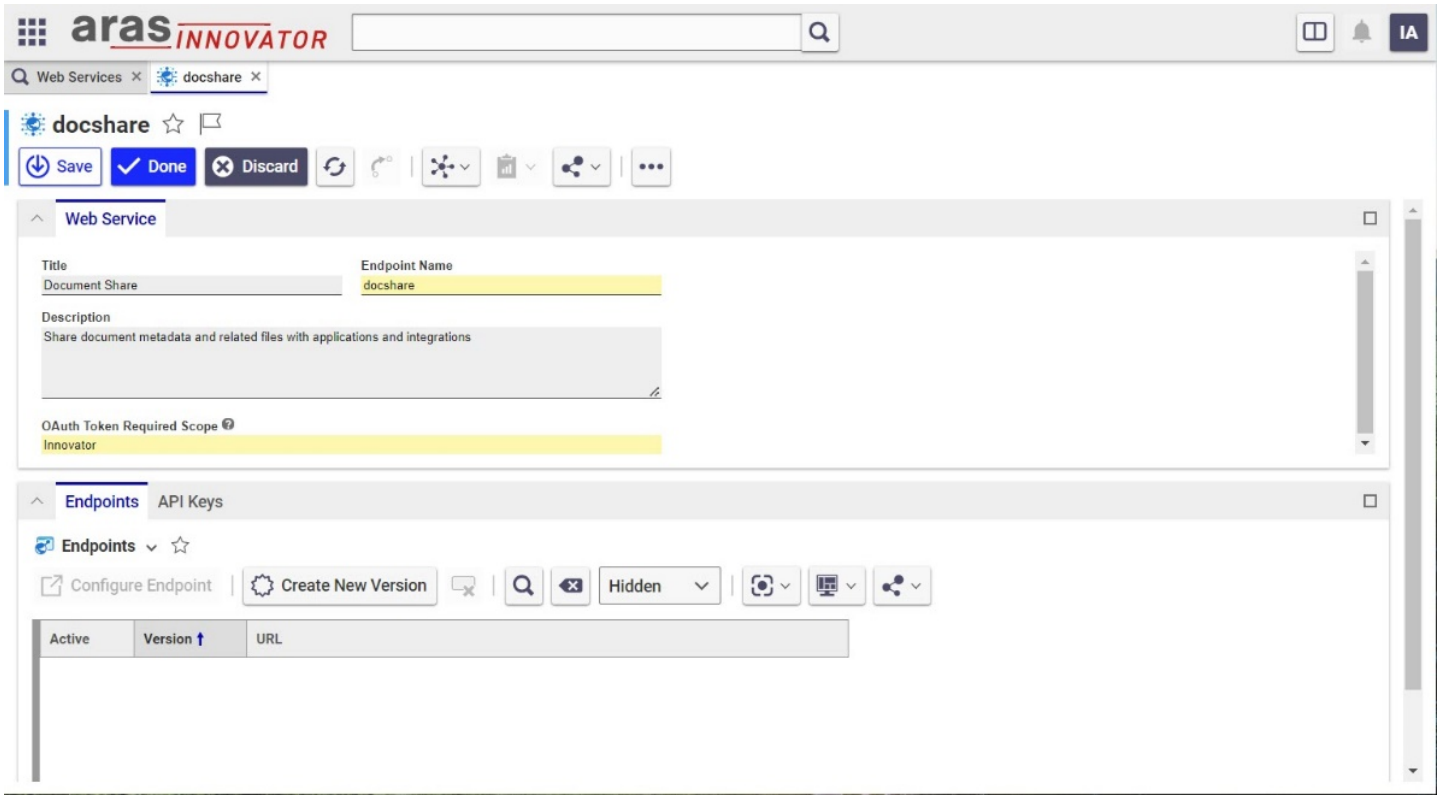


3. Click **Create New Web Service**.

   The **Web Service** form appears.



4. **Enter** the required value in the **Title**, **Endpoint Name,** and **Description** fields.

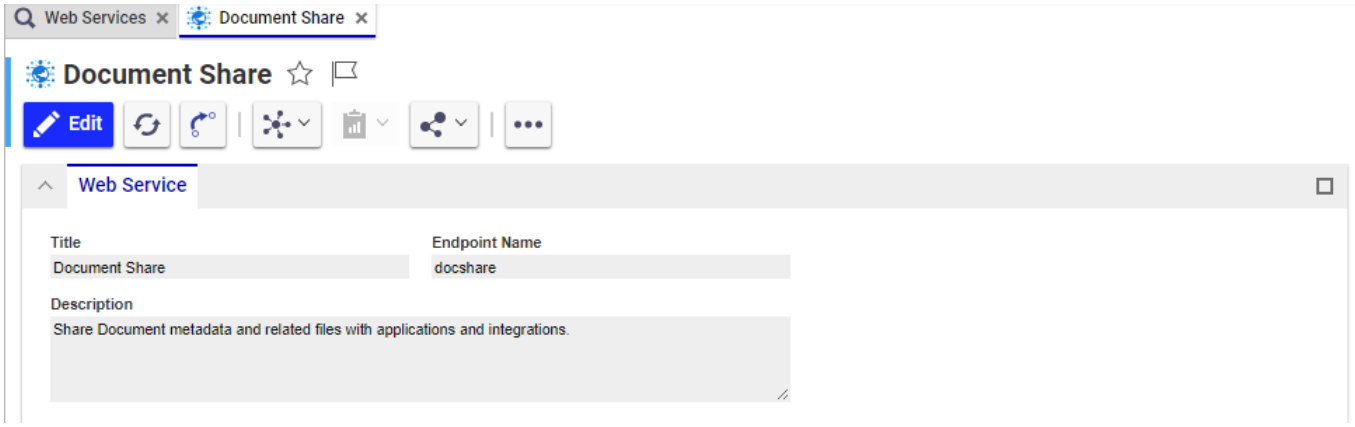


Important

The Endpoint Name property is used in the Web Service's URL. This string may include alphanumeric characters and underscores. Do not use spaces or other special characters.

5. The **OAuth Token Required Scope** property defines the scope(s) that must be present when authorizing a request to this web service using an OAuth Client. This setting is optional. The default value is "Innovator."
6. Click **Done** to save the changes.

7. The newly created **Web Service** form appears:



Creating an Endpoint

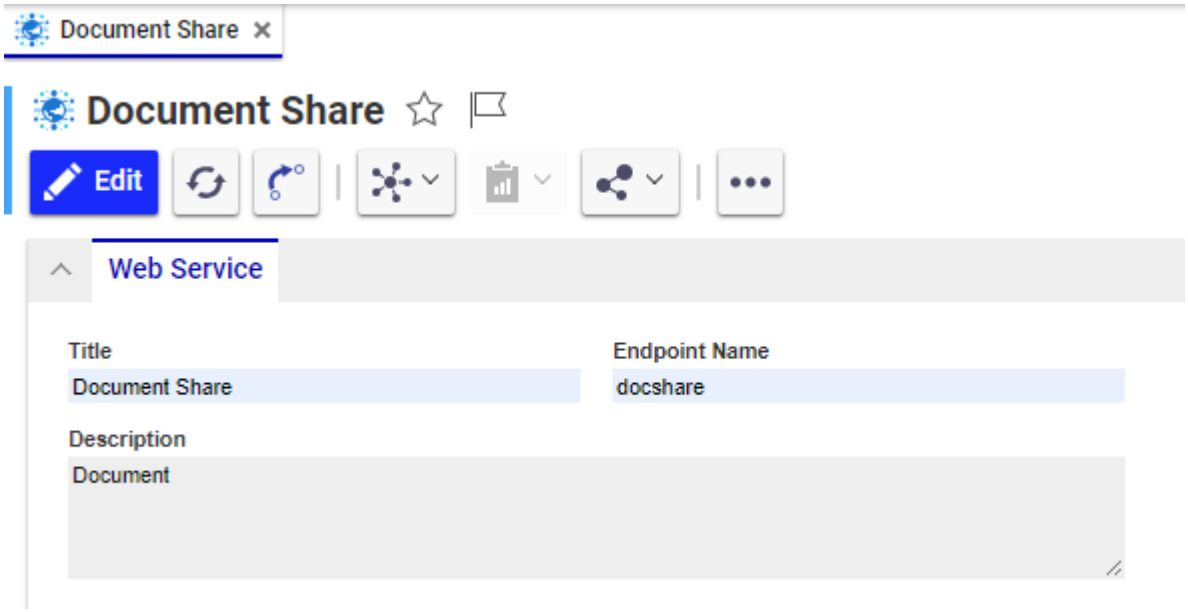
An Endpoint describes the scope of data, actions, and custom logic allowed for a Web Service. A Web Service must have at least one active Endpoint to "publish" a REST API. A Web Service may also have multiple Endpoints, each with a version number and unique URL. This capability enables users to make significant changes to an API without breaking existing client applications or integrations.

Important

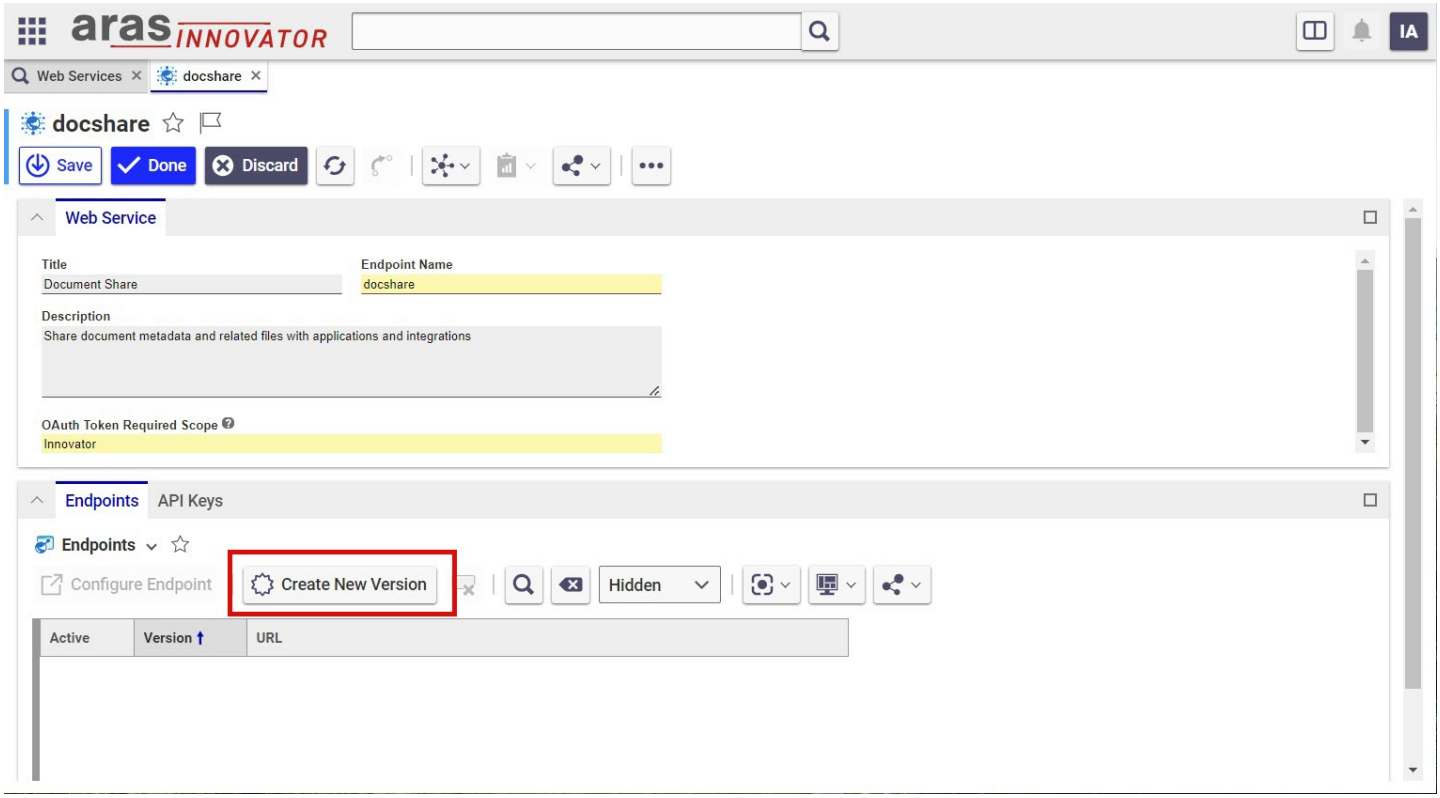
Because a REST API may have multiple active versions, Endpoints do not use the same behavior as other versionable items on the Aras Innovator platform. Refer to section Versioning a Web Service for more information about creating a new Web Service version.

The following steps outline the process to create a new Endpoint:

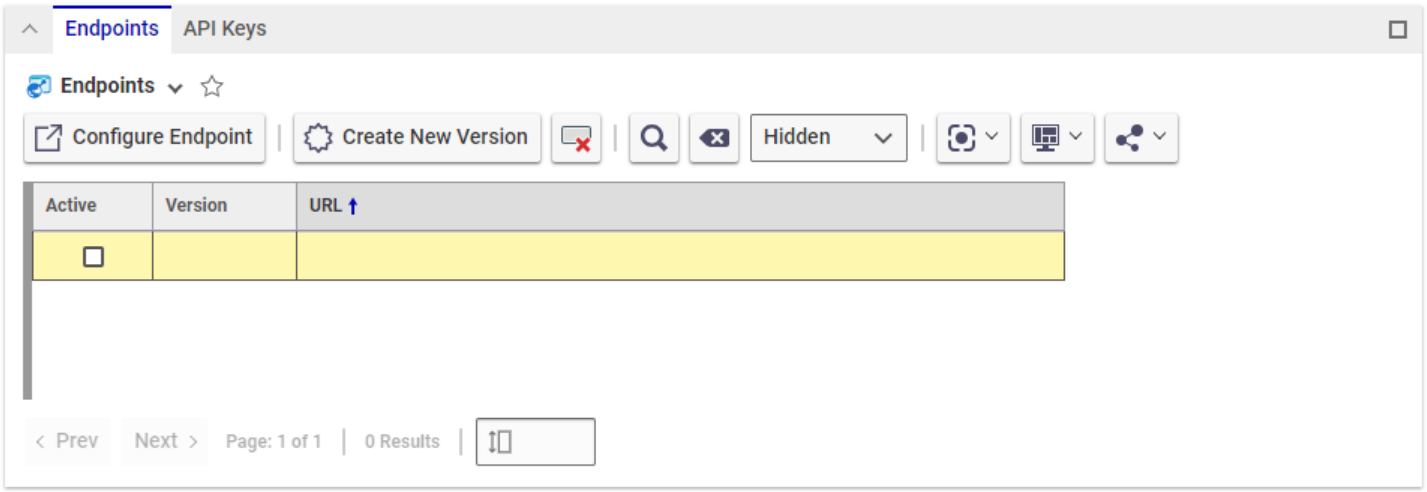
1. Open the **Web Service**.
2. Click **Edit**.



3. Click **Create New Version** in the **Endpoints** section.

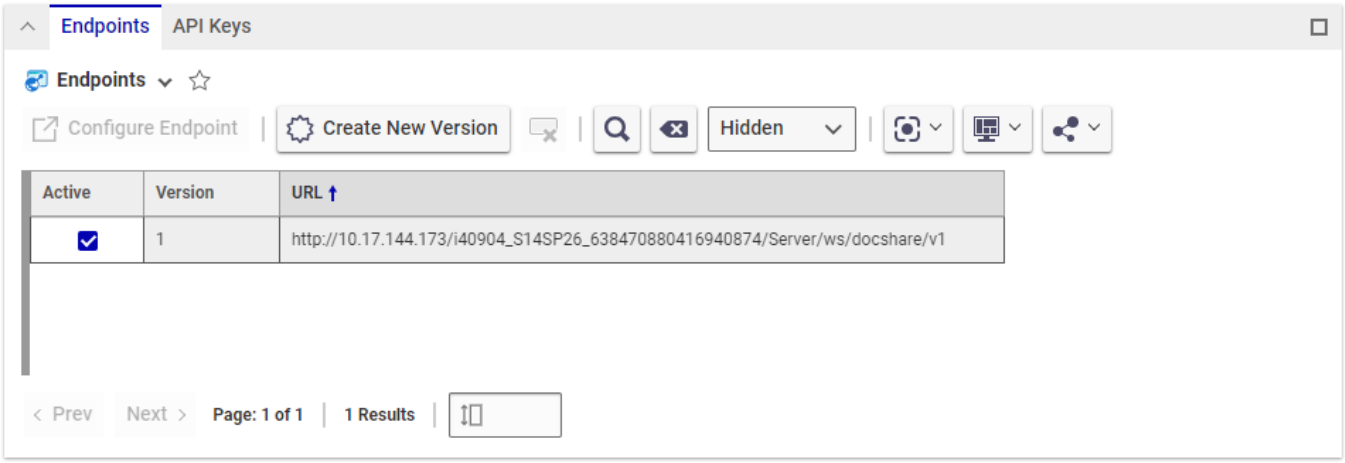


A row will be added to the **Endpoints** grid.

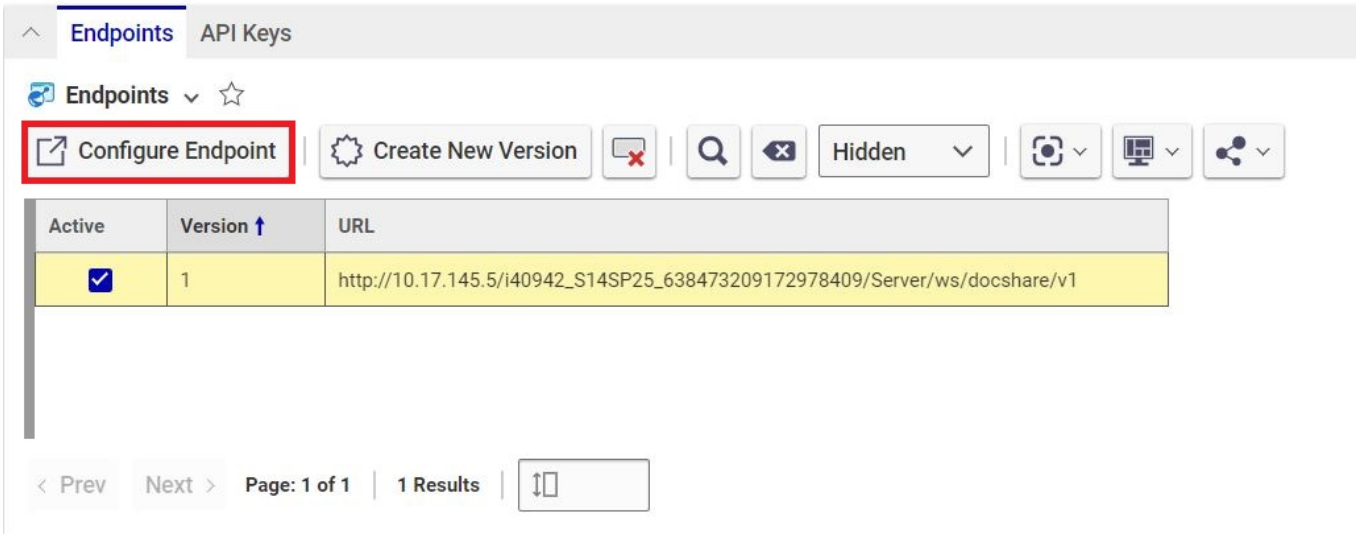


4. Click **Save**.
5. The Version and the endpoint URL will be updated automatically. Each endpoint has a unique URL with the following format:

   `{Innovator_url}/Server/ws/{endpoint_name}/v{endpoint_version}`



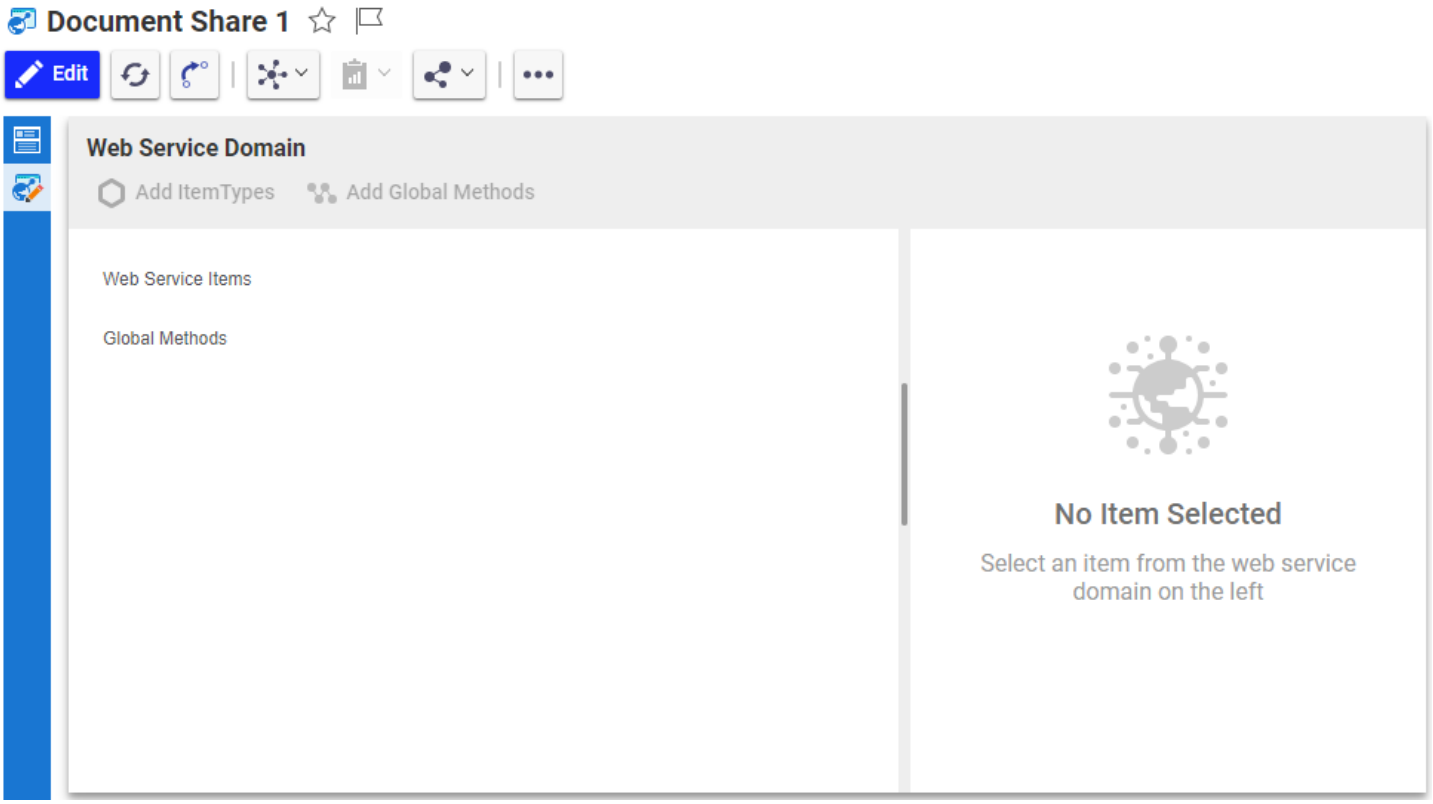
6. Select the **Endpoint**.
7. Click **Configure Endpoint**.



The Web Service Editor opens.

The Web Service Editor consists of the following sections:

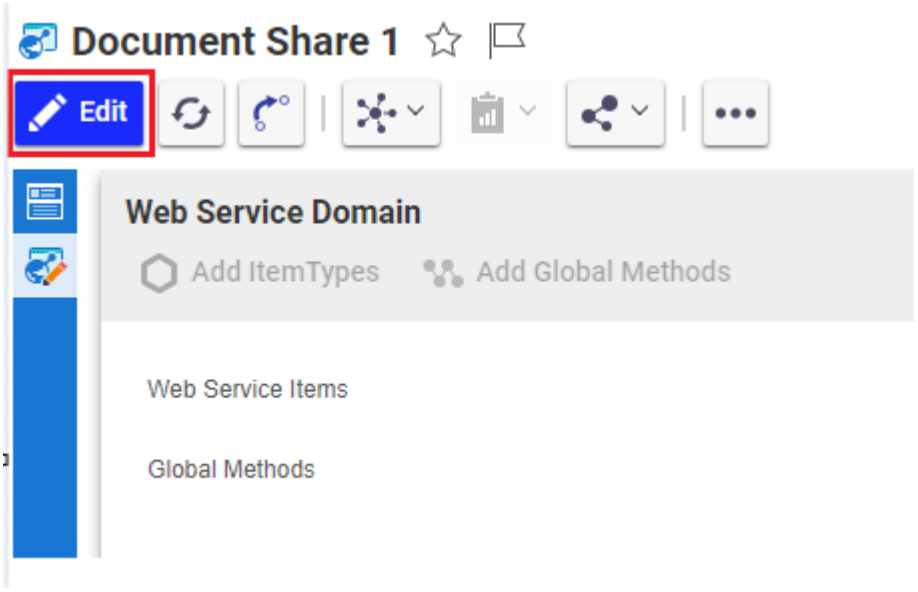
- The pane on the left lets the user view and modify the Web Service Items (ItemTypes) and Global Methods included in the endpoint's scope.
- The right pane enables the user to view and modify settings for any Web Service Item or Global Method selected in the left pane.

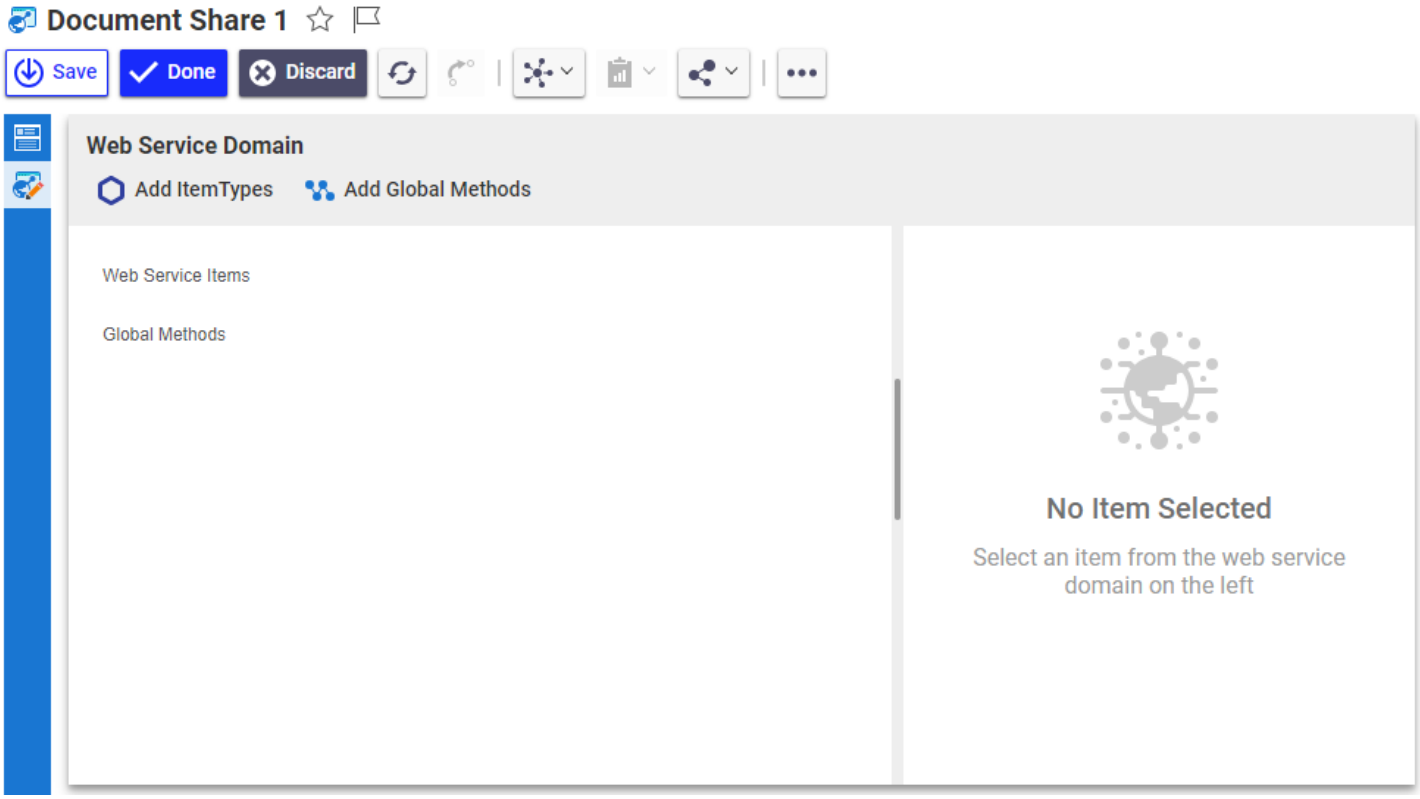


Adding ItemTypes

The following steps outline the process to add an ItemType to an Endpoint:

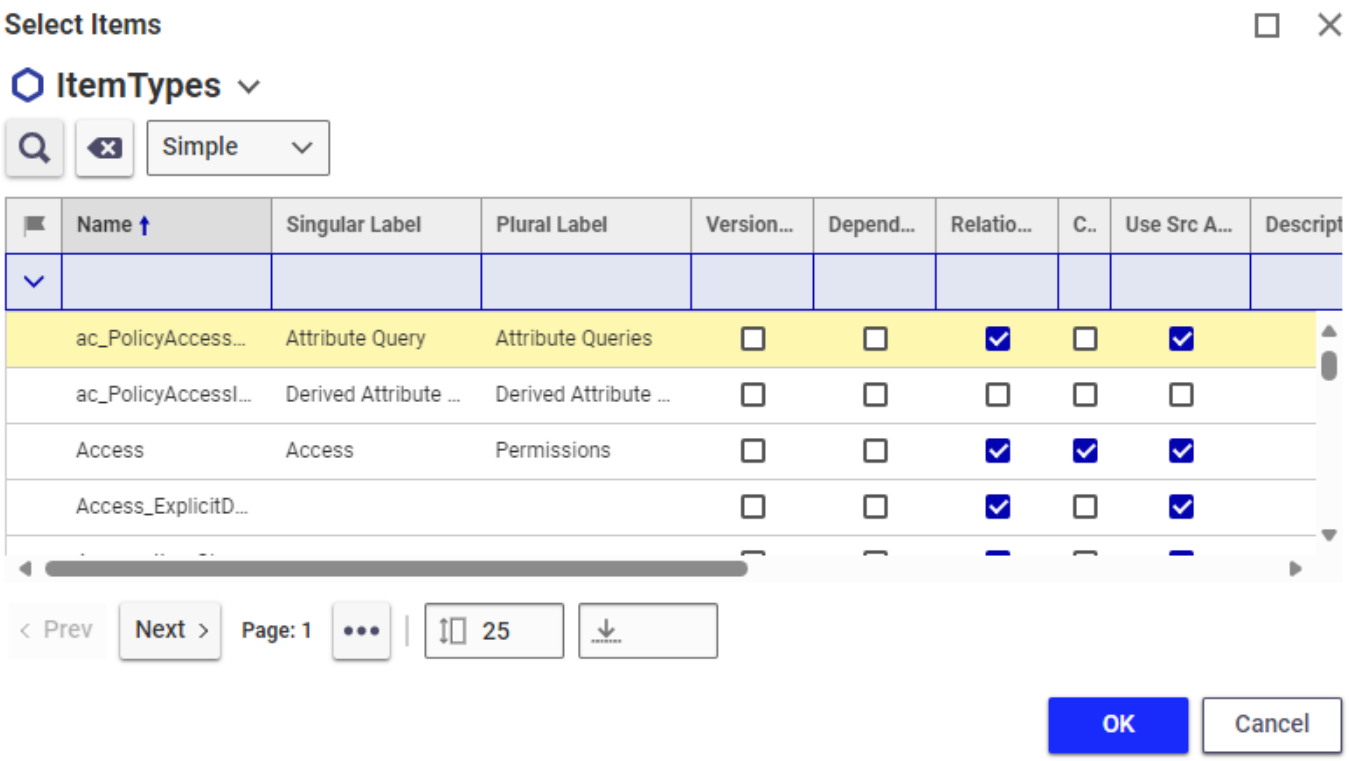
1. Click **Edit** in the **Web Service Editor**.





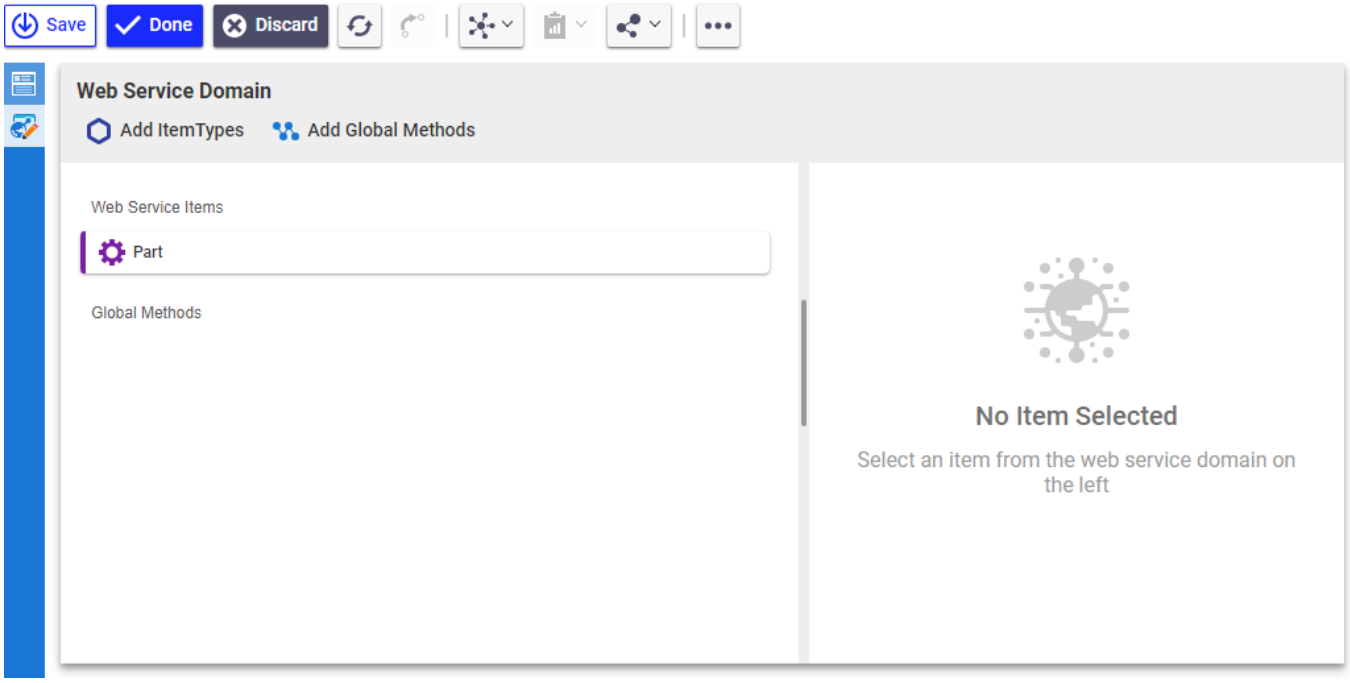
2. Click **Add ItemTypes**. The **Select Items** dialog box appears.

3. Click **Search**. A list of all the **ItemTypes** displays.



4. Select the required **ItemType(s)** and click **OK.**

   The selected **ItemType(s)** will be added to the **Web Service Items** list.



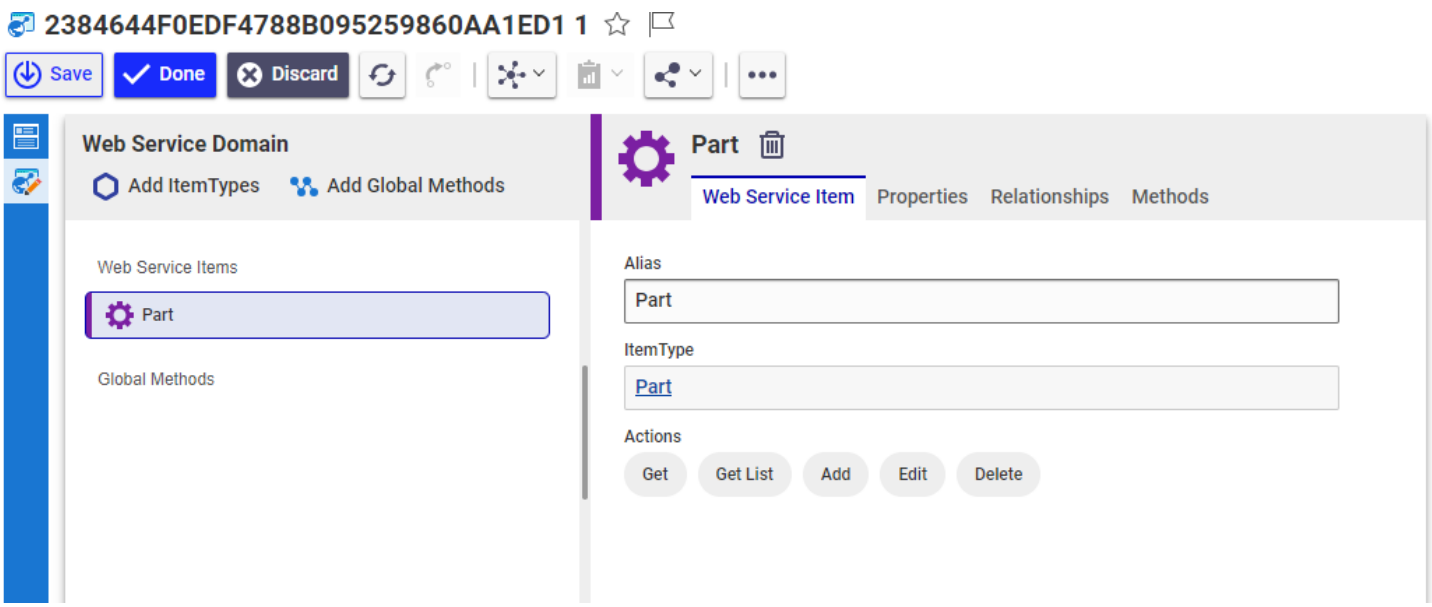
Web Service Items may be removed from the Endpoint by selecting the item in the left pane and clicking the Delete button in the header of the right pane. Removing a Web Service Item from an Endpoint does not delete the ItemType from the database.

Configuring a Web Service Item

The following steps outline the process to configure a Web Service Item.

1. Click on a **Web Service Item** in the left pane.
2. The right pane will display settings for the Web Service Item in four tabs:
    - **Web Service Item**: Default tab displayed when a Web Service Item is selected
    - **Properties**: Web Service Item's properties that are available via the Endpoint
    - **Relationships**: Web Service Item's relationships that are available via the Endpoint
    - **Methods**: Server Methods that can be run on the Web Service Item via the Endpoint

Web Service Item



1. (Optional) Enter an **Alias**.

   This string defines the name of the entity type representing the associated ItemType in the published API. The default value will be the name of the selected ItemType, and all spaces will be replaced with underscores.

   > **Important**
   >
   > Because Alias is used as the entity type in REST API calls, it must be unique and cannot contain spaces or special characters.

2. Select the Actions permitted for this Web Service Item. **Get:** Request item by id.

   - **Get List:** Request item(s) without the id parameter. Requires Get to be selected.
   - **Add:** Create a new item.
   - **Edit:** Edit an item.
   - **Delete:** Delete an item.

Important

Configurable Web Service obeys all access controls configured on the Aras Innovator server. The Action settings for Web Service Items allow admins to *permit or restrict* certain operations for an Endpoint. Enabling an Action on a Web Service Item *does not grant* additional permissions.

Properties

The **Properties** tab displays a table of properties included in the scope of the selected Web Service Item. The column definitions are as follows:

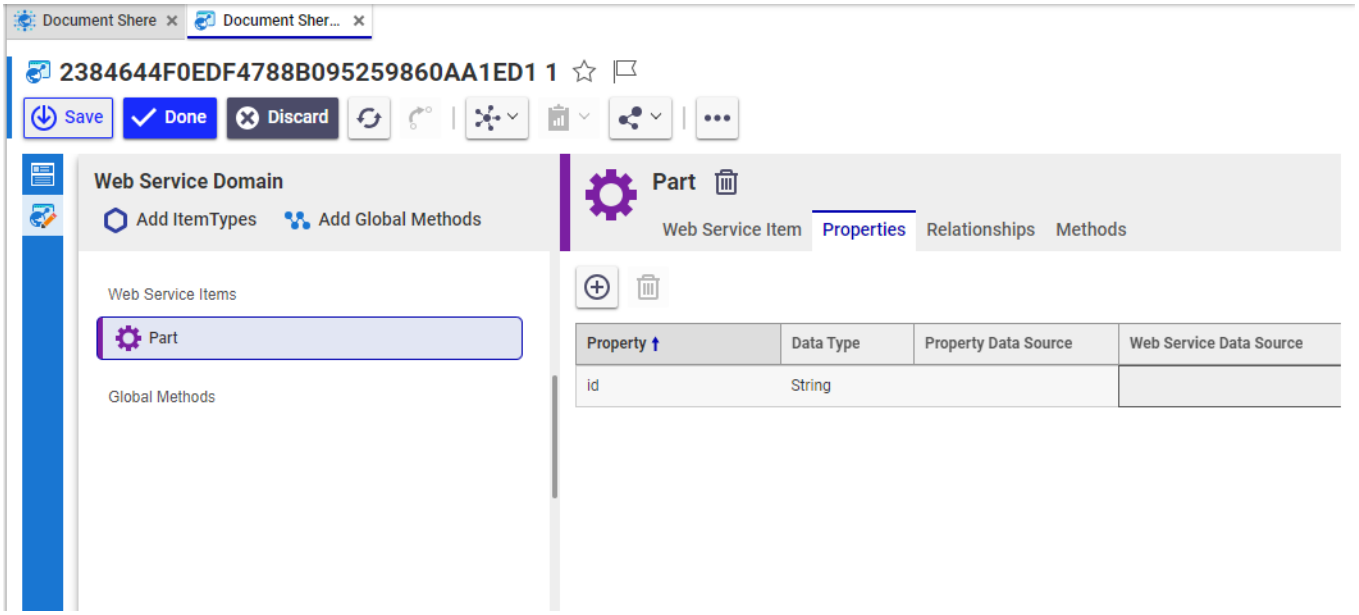
- **Property:** Displays the name of the property
- **Data Type:** Displays the property's data type as defined on the ItemType
- **Property Data Source:** Displays the property's data source as defined on the ItemType
- **Web Service Data Source:** For properties with the "Item" data type, this field identifies a Web Service Item associated with the property's Property Data Source. One will be created automatically if the Endpoint has no corresponding Web Service Item.

Important

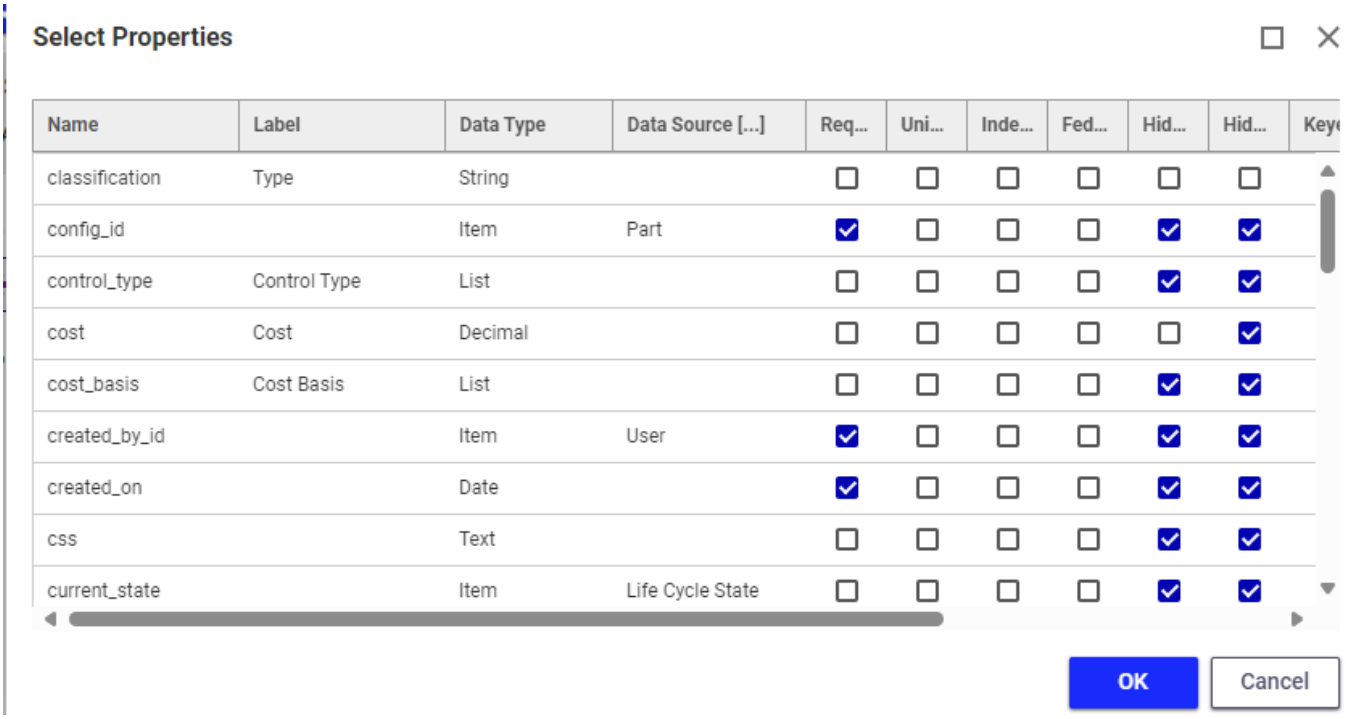
The id property is required for all Web Service Items and cannot be removed from the table.

The following steps outline the process to select properties for the Web Service Item:

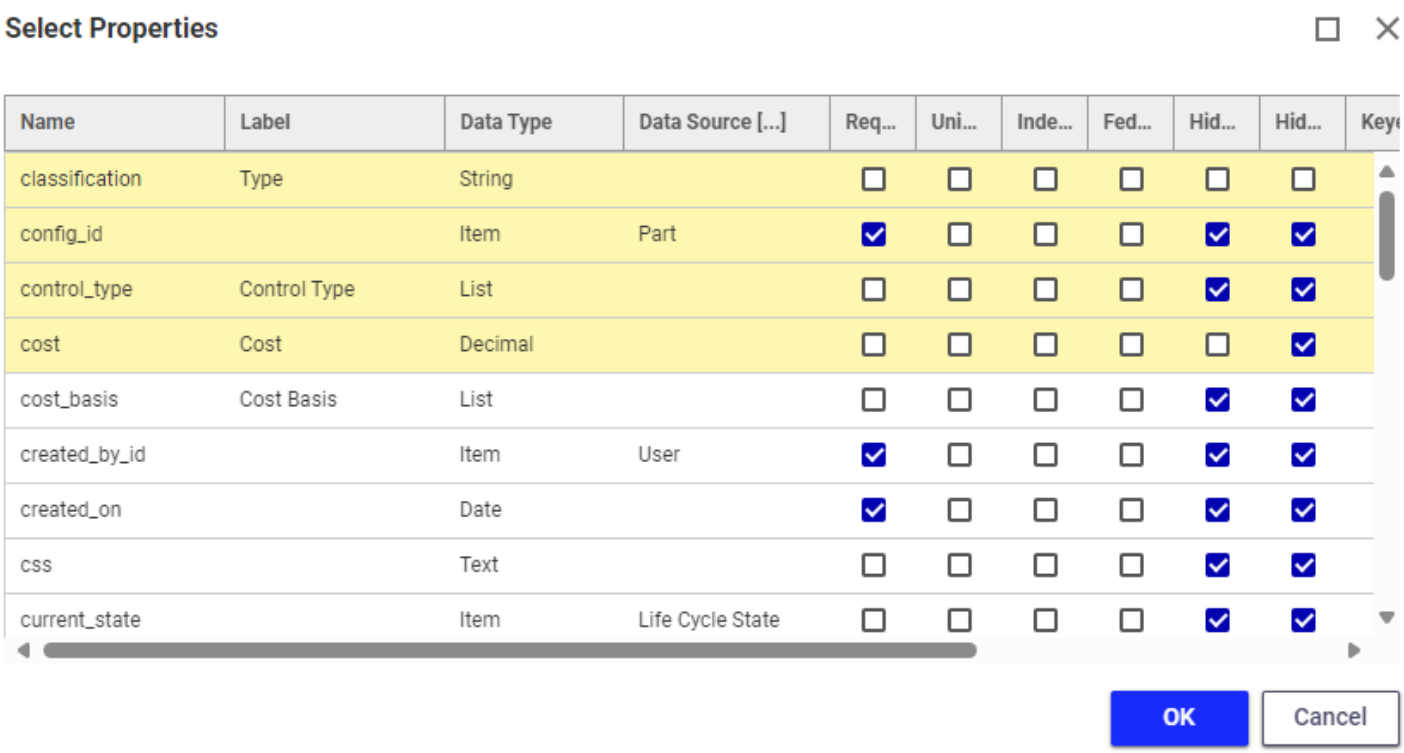
1. Click the **Properties** tab in the right pane.
2. Click **Add** in the Properties tab.



The **Select Properties** dialog appears.

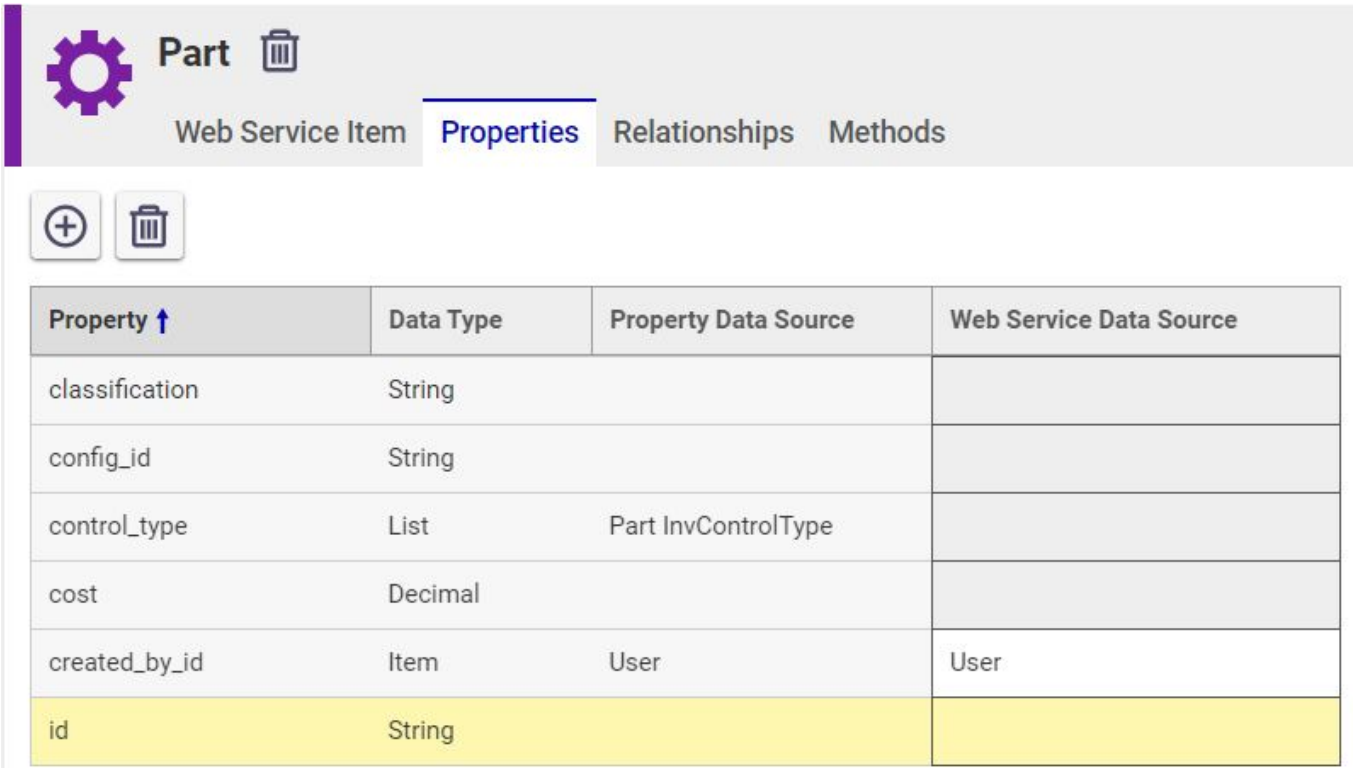


3. Select one or more properties.



5. Click **OK.**

The newly added properties will be displayed as shown below:



Property	Data Type	Property Data Source	Web Service Data Source
classification	String		
config_id	String		
control_type	List	Part InvControlType	
cost	Decimal		
created_by_id	Item	User	User
id	String		

Properties may be removed from the Web Service Item by selecting one or more properties and clicking the delete button directly above the table. Removing a property from a Web Service Item does not affect the ItemType definition.

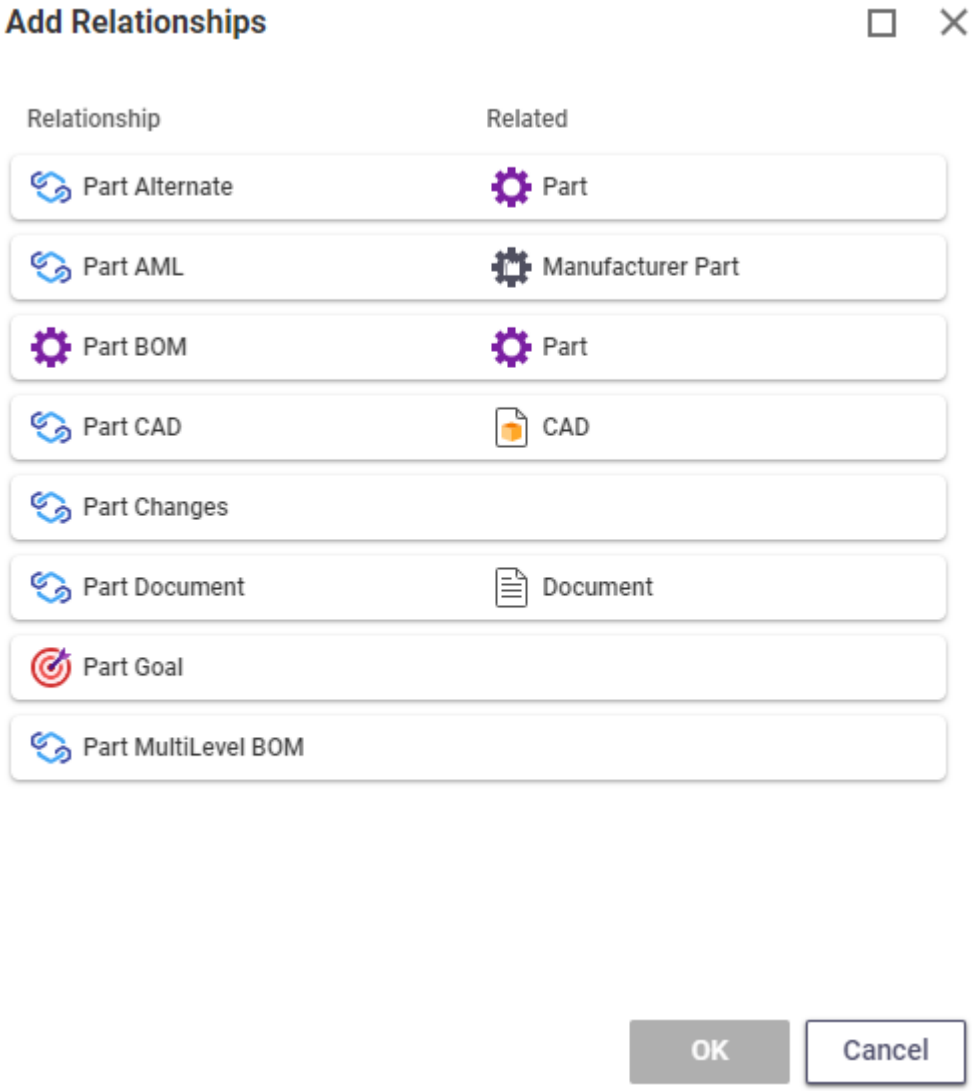
Relationships

The Relationships tab displays the Relationship Types included in the scope of the selected Web Service Item. The column definitions are as follows:

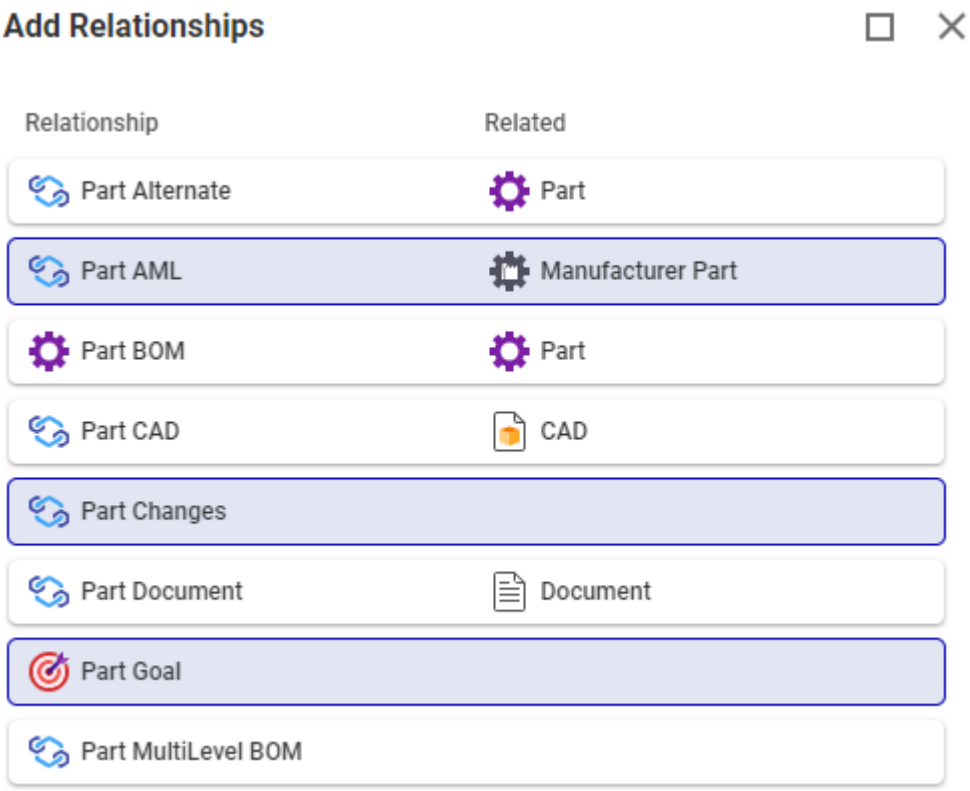
- **RelationshipAlias**: Name of the RelationshipType
- **RelatedAlias**: Identifies a Web Service Item associated with the relationship's ItemType. One will be created automatically if the Endpoint has no corresponding Web Service Item.

1. Click the **Relationships** tab.
2. Click **Add** in the Relationships tab.

The **Add Relationships** dialog appears. It lists the Relationships for the Web Service Item's associated ItemType and the related ItemType.

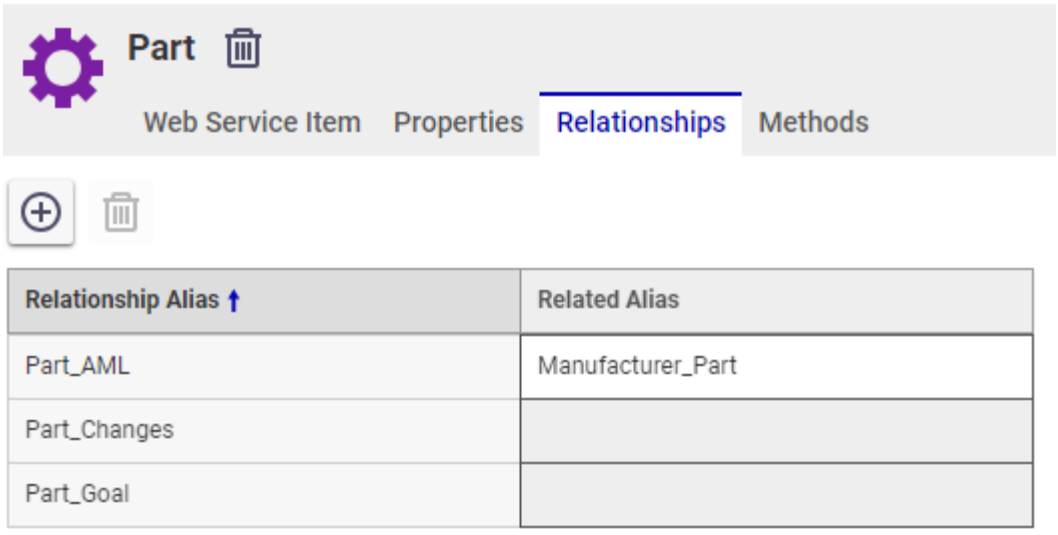


3. Select one or more Relationships.



4. Click **OK**.

   The newly added relationships will be displayed in the table.

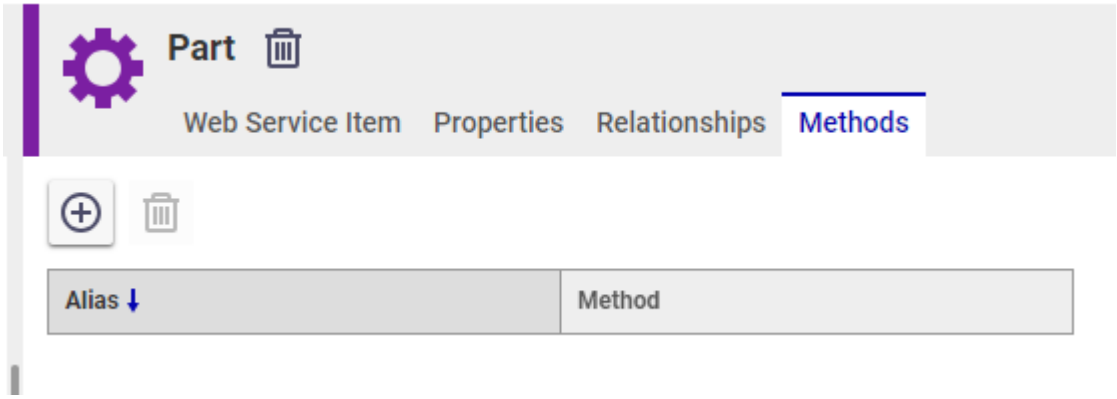


Relationships may be removed from the Web Service Item by selecting one or more rows and clicking the delete button directly above the table. Removing a relationship from a Web Service Item does not affect the ItemType definition.

Methods

The **Methods** tab displays the item methods, or “bound methods,” included in the scope of the selected Web Service Item. These should be server Methods explicitly developed for a context item of this ItemType.

> **Important**
>
> All Server Events configured on the ItemType will function as expected for events triggered by REST requests. Those Methods must only be added to the Web Service Item if they should be callable through the Endpoint.



The column definitions are as follows:

- **Alias**: The name used to execute the Method via the Endpoint
- **Method**: The server Method that will be executed

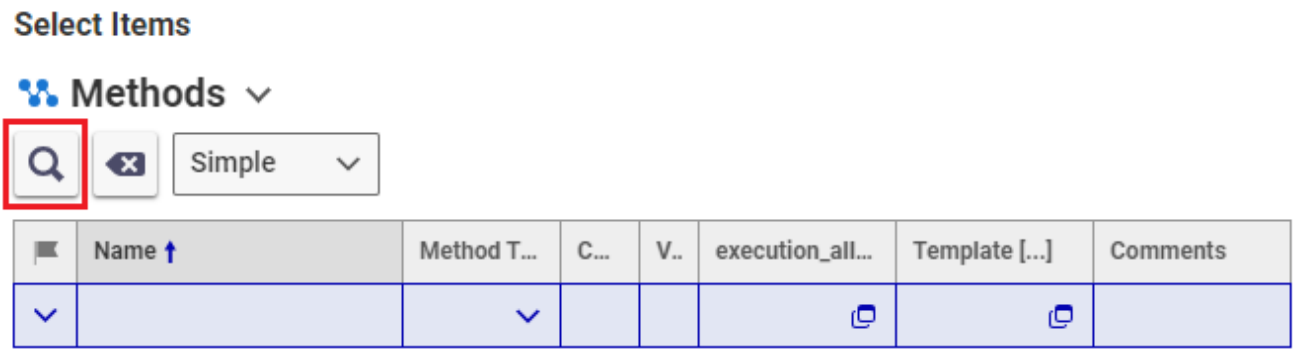
> **Important**
>
> Because the Method Alias will be used in REST API calls, it must be unique and cannot contain spaces or special characters.

The following steps outline the process of adding Methods:

1. Click the **Methods** tab.

2. Click **Add** in the Methods tab. The **Select Item** dialog appears.

3. Click **Run Search**.

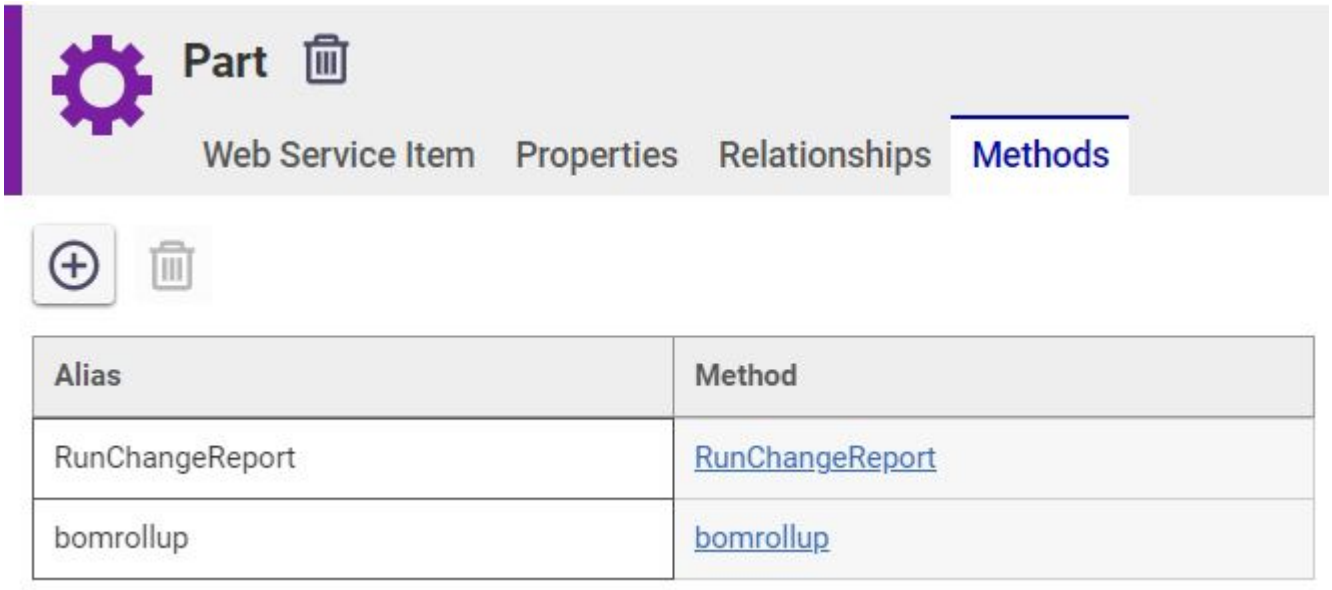


A list of items will be displayed.

4. Select one or more items.

5. Click **OK**.

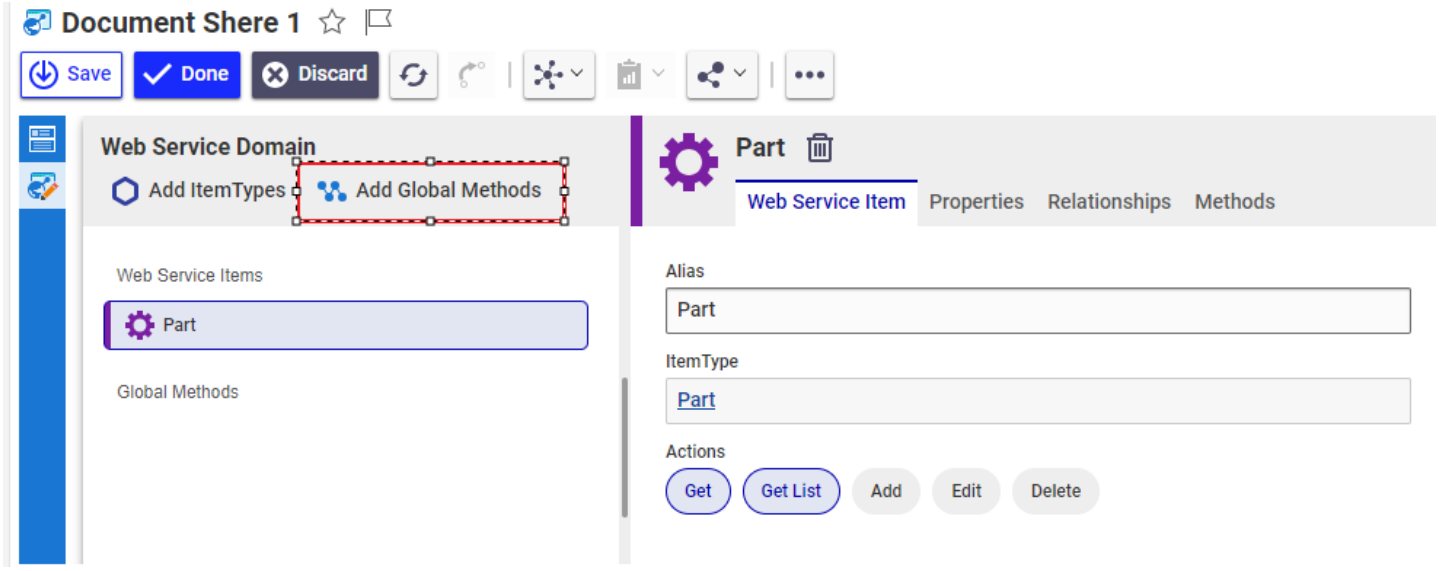
The newly added records will be displayed.



Methods may be removed from the Web Service Item by selecting one or more rows and clicking the delete button directly above the table. Removing a Method from a Web Service Item does not delete the Method from the database or affect the ItemType definition.

Adding Global Methods

Global Methods, or “unbound methods”, are server Methods that can be executed independently of a specific context item type.



The column definitions are as follows:

- **Alias**: The name used to execute the Method via the Endpoint
- **Method**: The server Method that will be executed

Important

Because the Method Alias will be used in REST API calls, it must be unique and cannot contain spaces or special characters.

The following steps outline the process to add Global Methods:

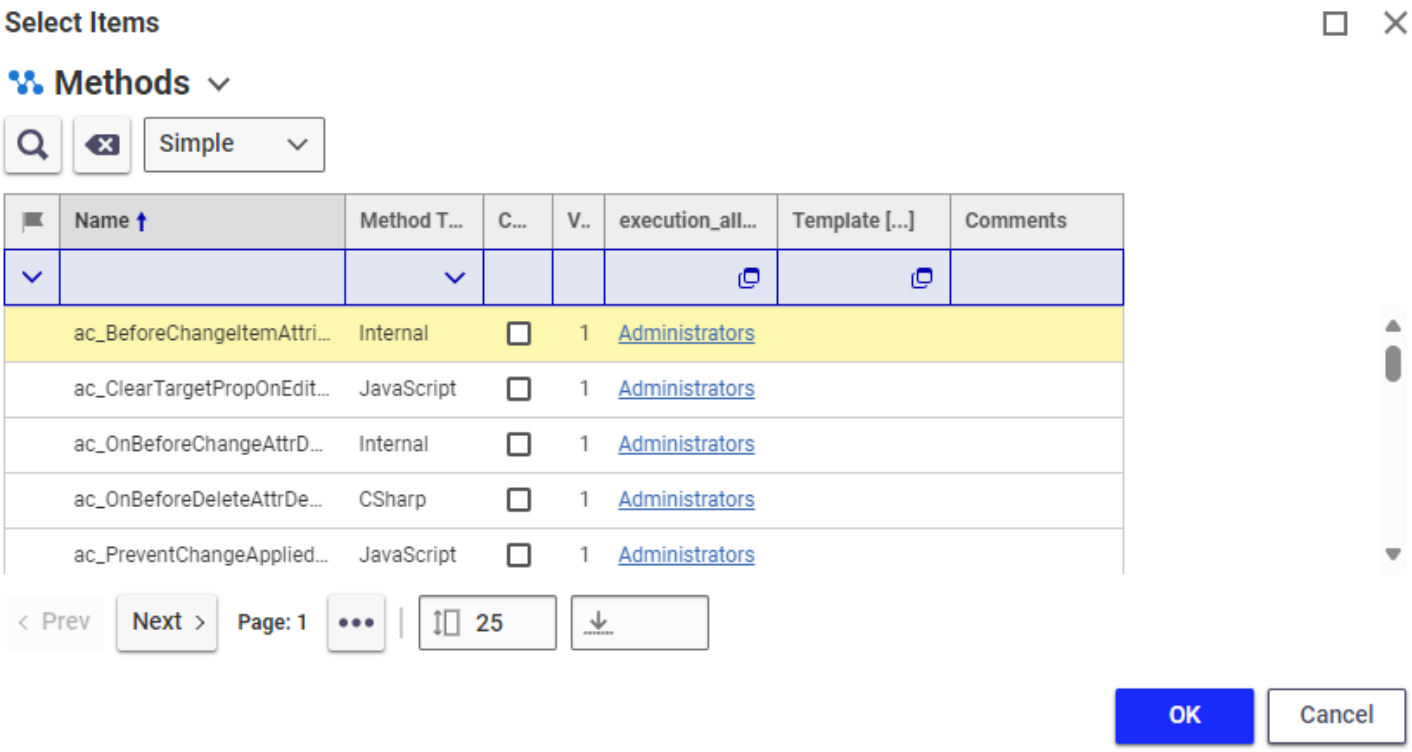
1. Click **AddGlobalMethods** directly above the left pane.

   The **Select Item** dialog appears.

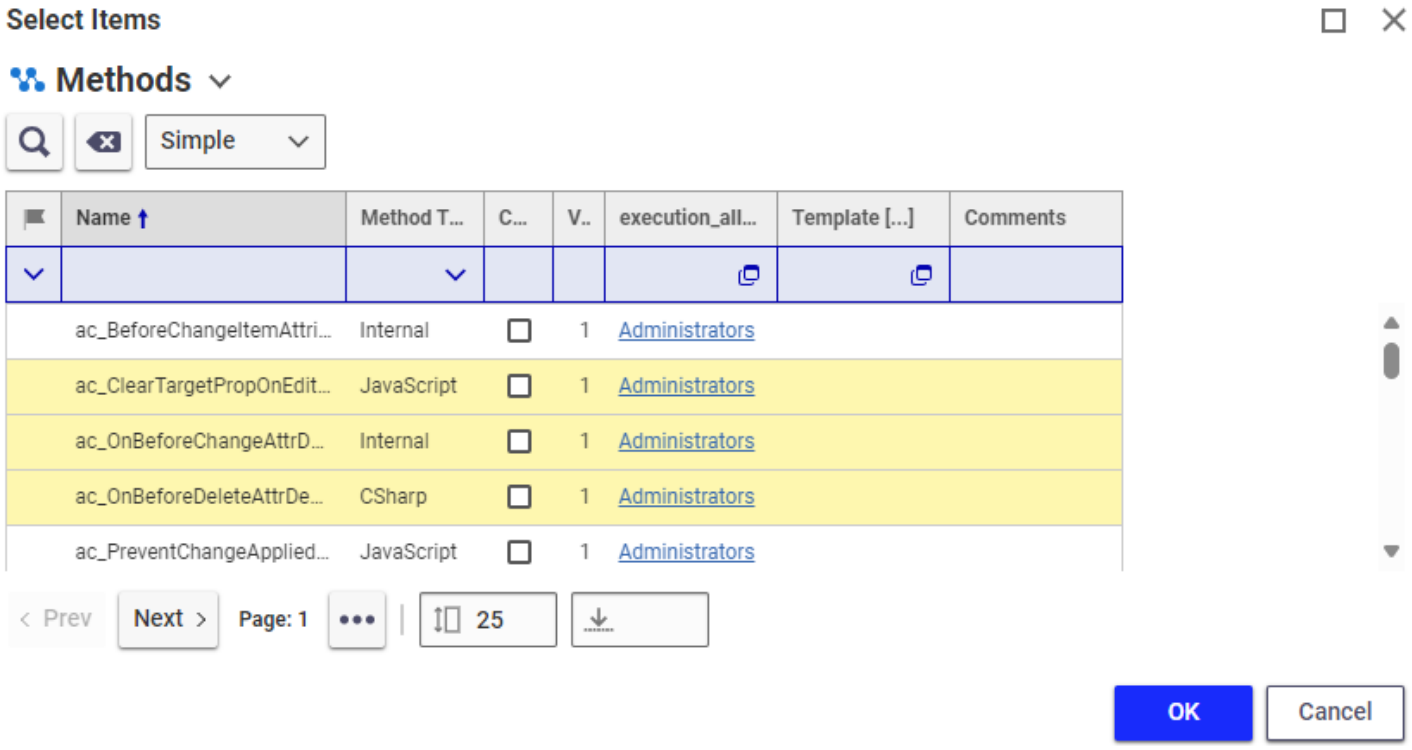
2. Click **Run Search**.

3. Click **Search**.

   A list of all the corresponding Items will be displayed.

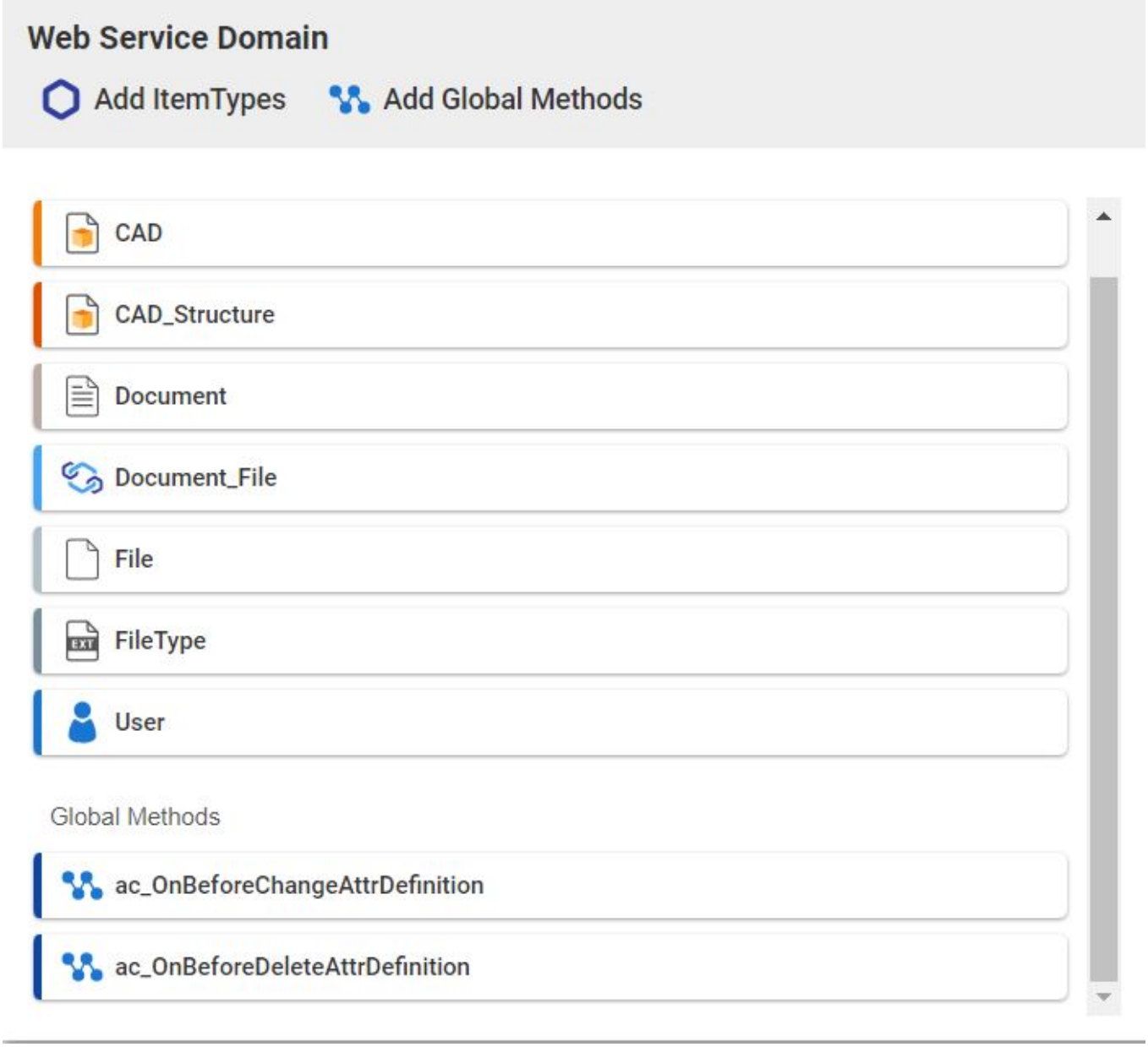


4. Select one or more server Methods.



5. Click **OK**.

   The newly added method(s) are in the left pane.



6. Click a **GlobalMethod** in the left pane.
7. The right pane will display the **Global Method**'s settings in a single tab.

- **Alias**: The name used to execute the Method via the Endpoint

- **Method**: The server Method that will be executed
- **Parameters**: This table enables admins to define named parameters that may be passed in the body of a request for use in the server Method code

> **Important**
>
> Because the Method Alias will be used in REST API calls, it must be unique and cannot contain spaces or special characters.

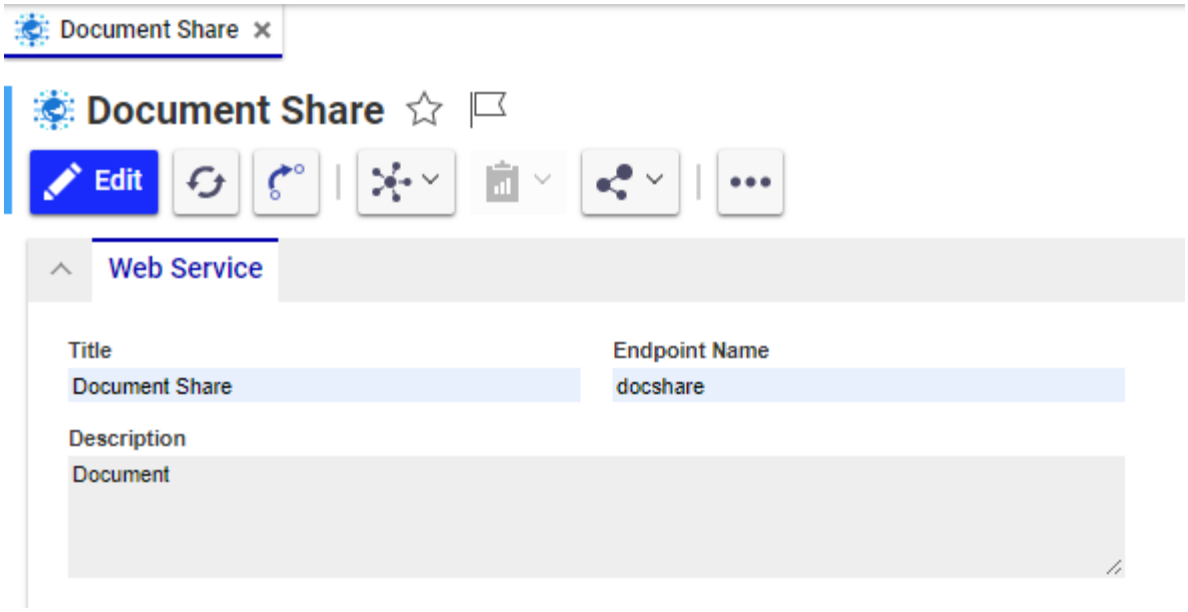
Global Methods may be removed from the Endpoint by selecting the Method in the left pane and clicking the **Delete** button in the header of the right pane. Removing a Global Method from an Endpoint does not delete the Method from the database.

8. Click **Done** to save the Endpoint.

Editing an Endpoint

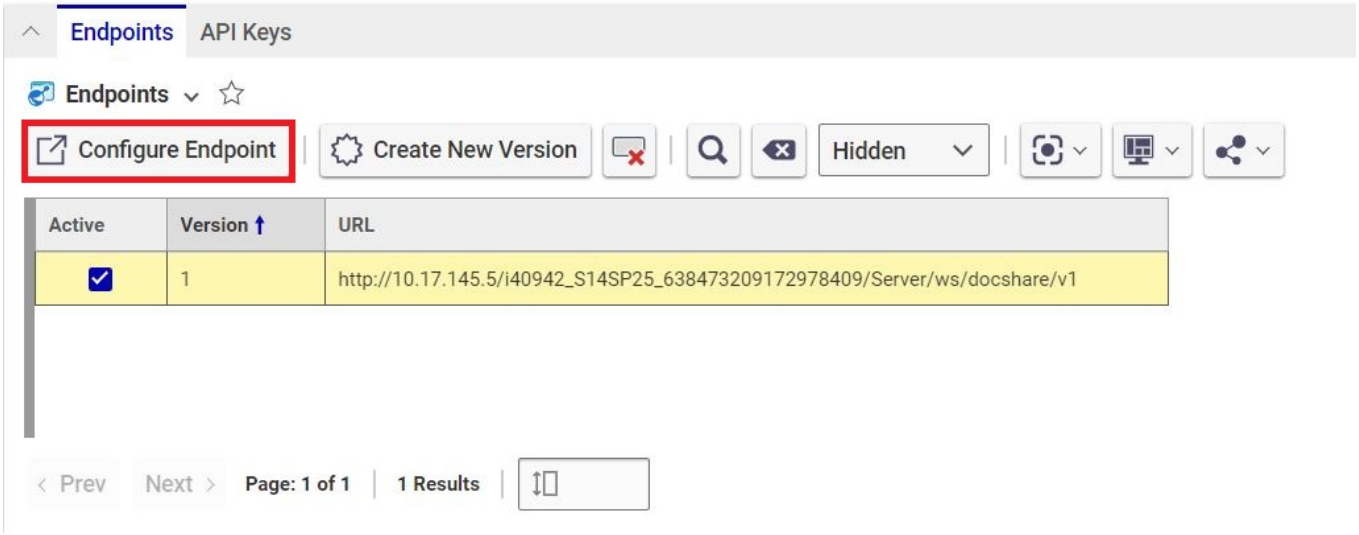
The following steps outline the process to create a new Endpoint:

1. Open the **Web Service**.
2. Click **Edit**.

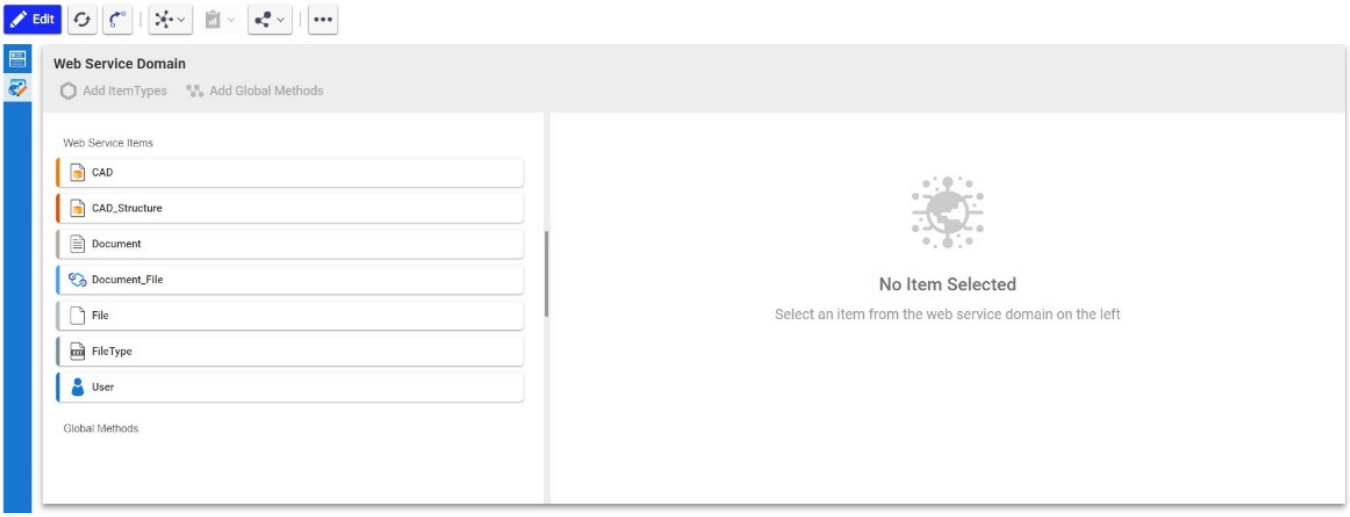


3. Select the Endpoint by clicking on it.

4. Click **Configure Endpoint**.



The Endpoint opens in the **Web Service** Editor.

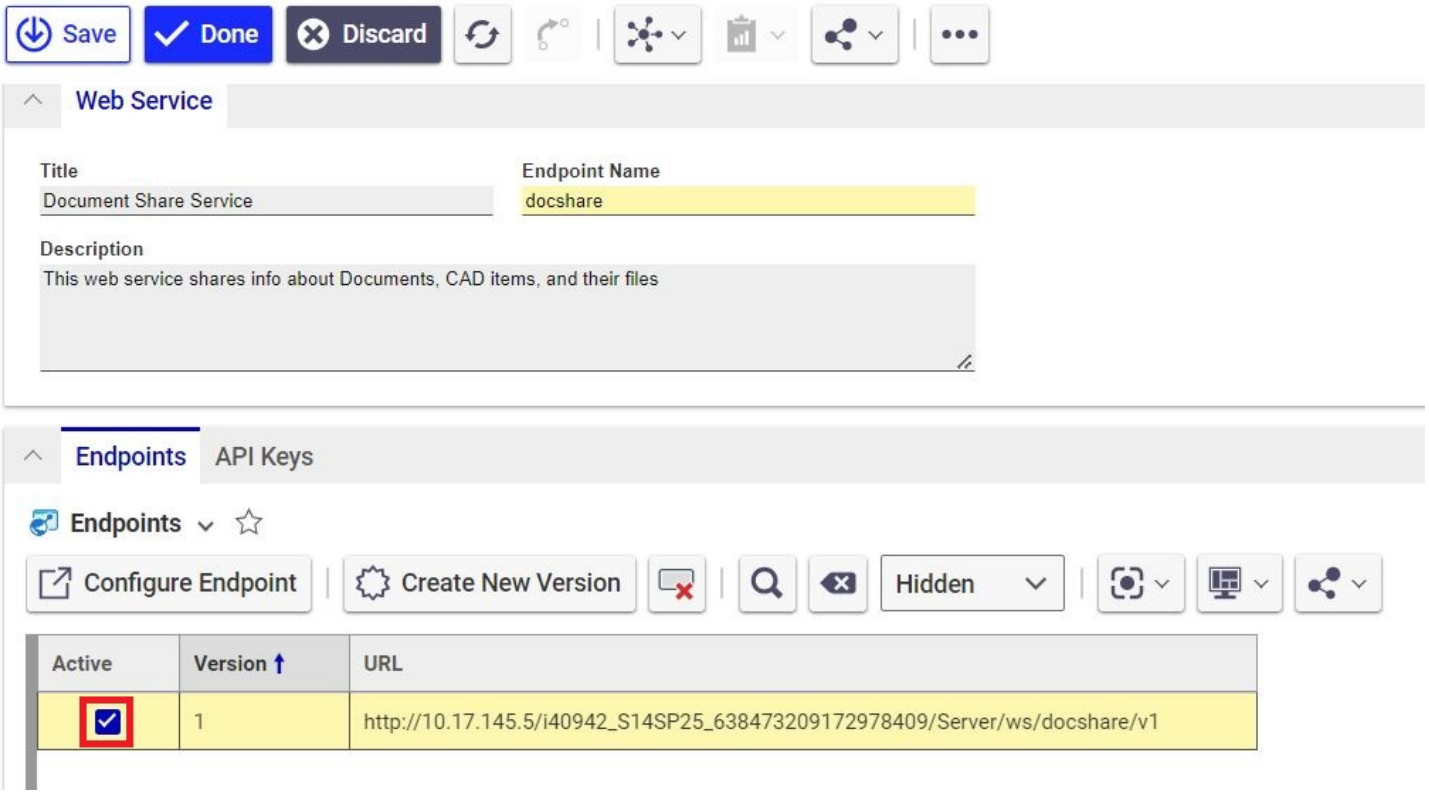


5. Click **Edit** in the Web Service Editor.
6. Edit the **Endpoint** as needed, following the appropriate steps from Sections **Adding ItemTypes** and **Configuring a Web Service Item**.
7. Click **Done** in the Web Service Editor when all edits are complete.

Publishing an Endpoint

Creating an Endpoint does not automatically make the REST API accessible. It must be published or made “active” before it can respond to requests. The following steps outline the process for publishing an Endpoint.

1. Open the Web Service.
2. Click **Edit**.
3. In the Endpoints grid, enable the Active checkbox for the Endpoint to be published.



4. Click **Done** in the Web Service toolbar.

   The Endpoint is now published and can respond to HTTP requests. The Endpoint can be unpublished again by disabling the Active checkbox.

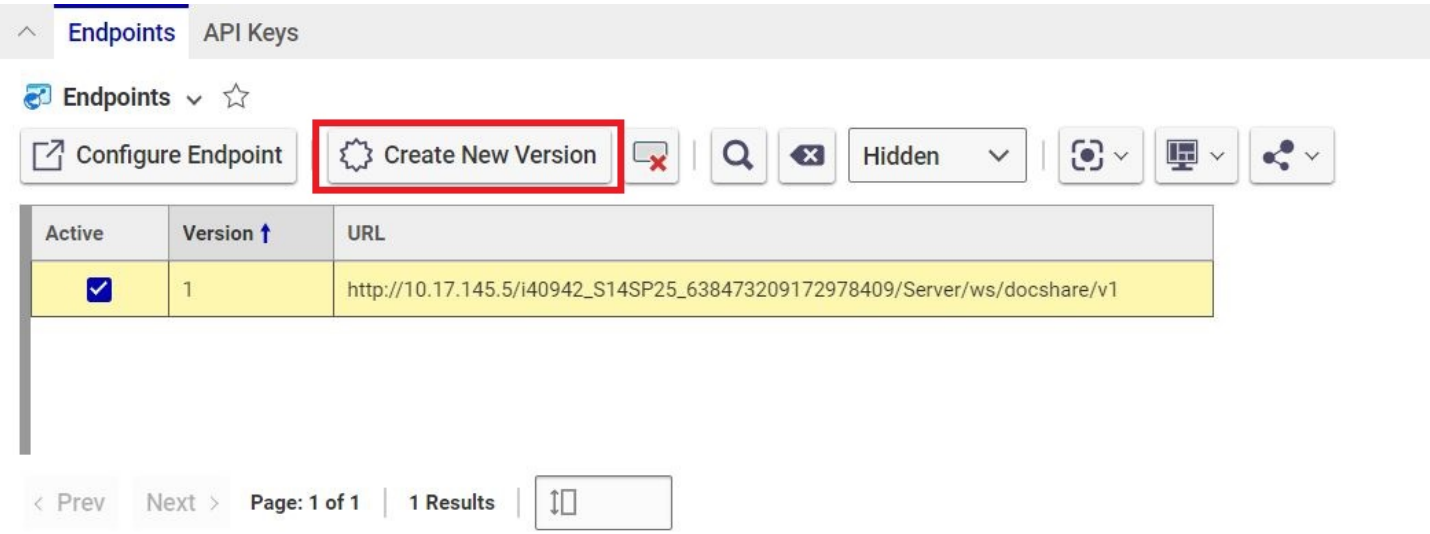
Important

The Active property can also be managed on the Endpoint form.

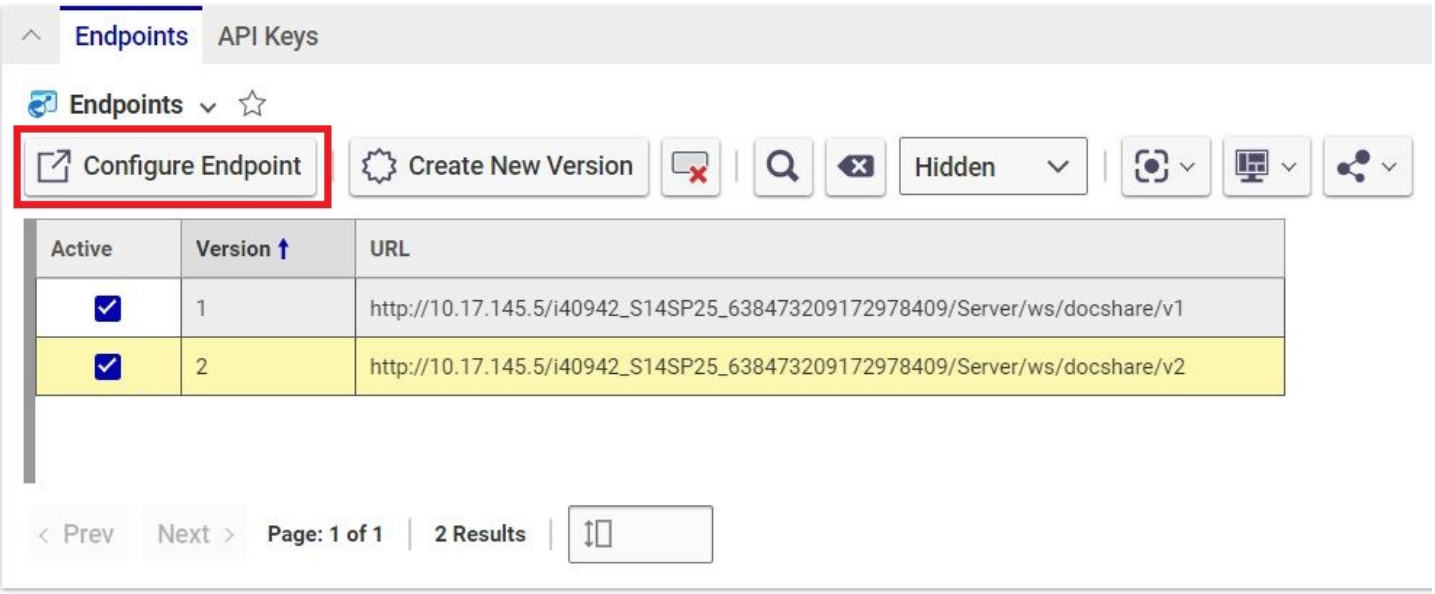
Versioning a Web Service

A Web Service may have multiple Endpoints, each describing a version of the Web Service. This enables the user to make significant changes to an API without breaking existing client applications or integrations. The following steps outline the process of versioning a Web Service with a new Endpoint.

1. Open the **Web Service**.
2. Click **Edit**.
3. In the Endpoints grid, select the Endpoint that will be the basis of the new version.
4. Click the **Create New Version** button in the grid toolbar.



4. Click **Save** in the Web Service toolbar.
5. In the **Endpoints** grid, select the **Endpoint** with the new version.
6. Click **Configure Endpoint** in the grid toolbar.



7. Edit the Endpoint as needed. Then Save the Web Service and publish the newest Endpoint version.

Important

A Web Service may have more than one published or “active” Endpoint, each with a unique url.

Configuring Authorization for a Web Service

Creating an API Key

An API Key enables the Aras Innovator server to authorize requests and execute the request as the User associated with the key. API Keys are most commonly assigned to a service User that is used for server-to-server requests or other scenarios where an end user cannot be prompted for credentials.

Important

Applications and integrations that execute requests as a specific end user should use the OAuth credential flow instead of API Keys. This is the same authorization approach supported by the platform’s default REST API.

API Keys contain the following properties:

Name: Name of the API Key

- **User**: The User that will be used for all requests made with this key. If a user is not entered, a system user will be automatically generated.
- **Description**: Description of the API Key
- **Created By**: The User who created the API key
- **Created On**: When the API key was created
- **Scope**: Optional. Selecting “Override System Properties” will enable requests using this API Key to overwrite the data in system properties.

Important

Only enable the “Override System Properties” scope when a use case requires an application or integration to overwrite the system property values in the Aras Innovator database. This feature may replace critical metadata such as created_by_id, modified_by_id, created_on, modified_on, state, etc. See the section “Overwriting System Properties” below for more detail on this capability.

Important

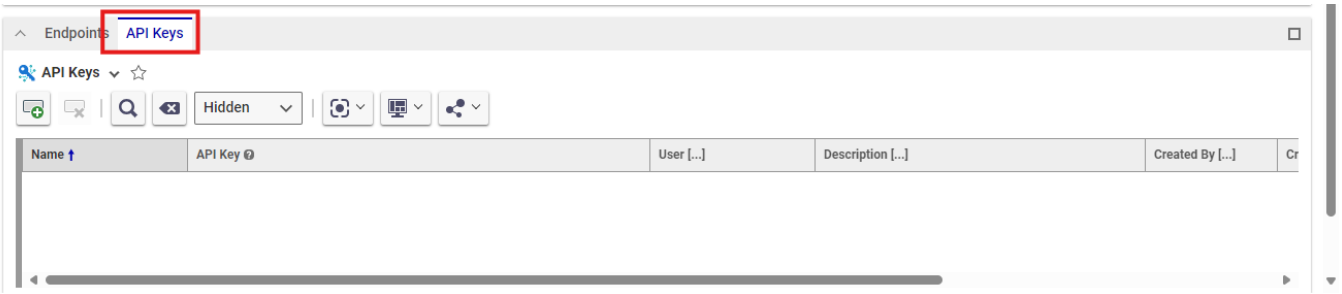
After generating the API Key, copy and store it securely, as it will not be visible again.

Important

System Users generated for API Keys are not automatically added to any group identities or access control configurations. Be sure to check that generated system users have permission to perform the operations described in the Endpoint.

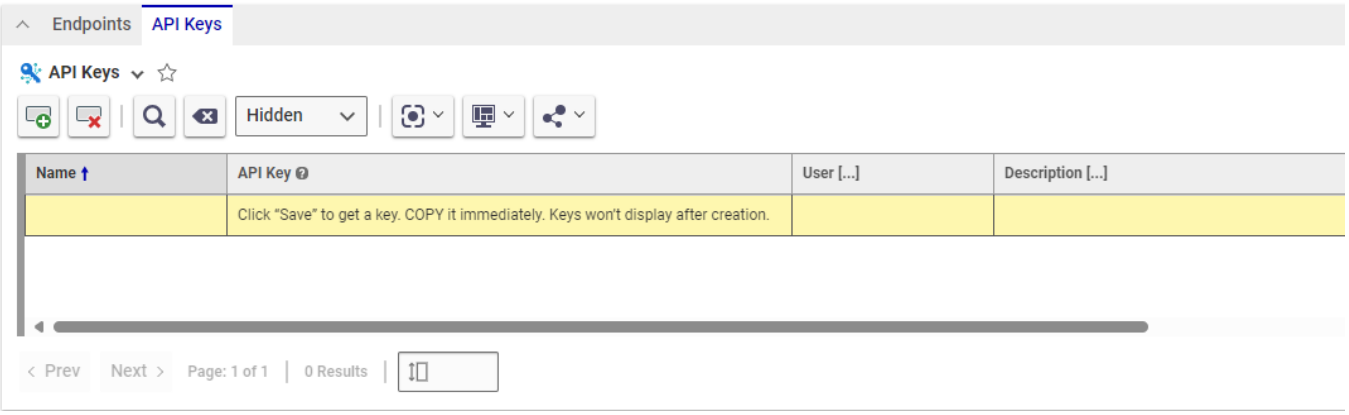
The following steps outline creating an API Key for a Web Service.

1. Open the Web Service.
2. Click **Edit**.
3. Click the **API Keys** tab.



4. Click the **New API Key** button.

   A new record will be added to the table, as shown below:



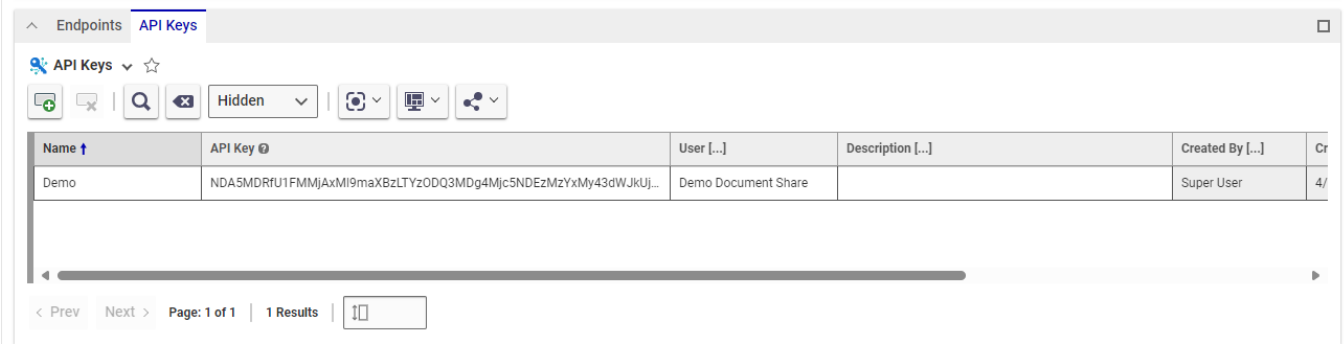
5. Enter **Name**.

6. (Optional) Enter the **User** that will be used for all requests made with this API Key.

   If a User is not selected, one will be generated. If using a generated user, be sure to grant that user permission for the requests that will be made with the API Key.

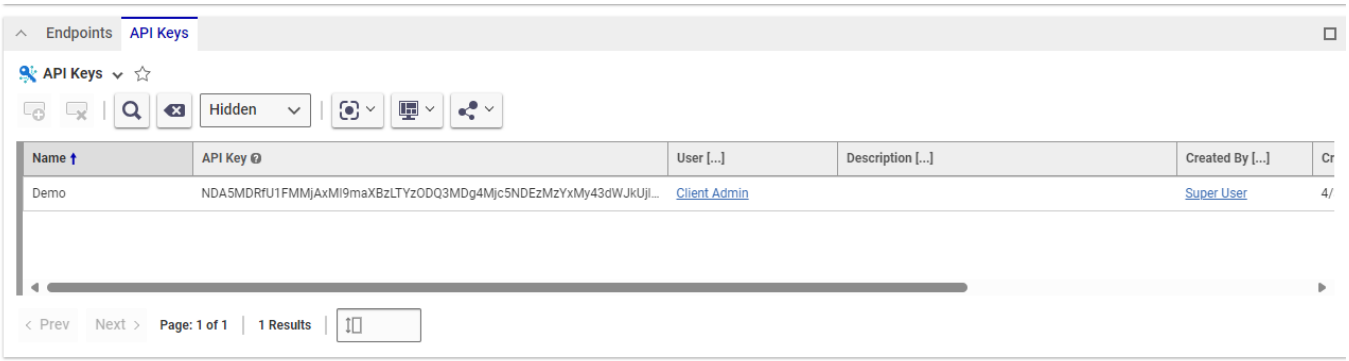
7. Click **Save**.

   Saving the record will display the API key. Save the key, as it will not be visible after creation.



8. Click **Done**.

9. The newly added **API key** will be displayed.



Using a Web Service

The Configurable Web Service produces REST APIs that follow the same OData protocol used by the platform's default REST API. As a result, sending requests and receiving responses from a CWS Endpoint is very similar to the default REST API. This section will outline steps that are unique to using a CWS Endpoint and provide references to the RESTful API documentation for steps shared with the default API.

Request URL Format

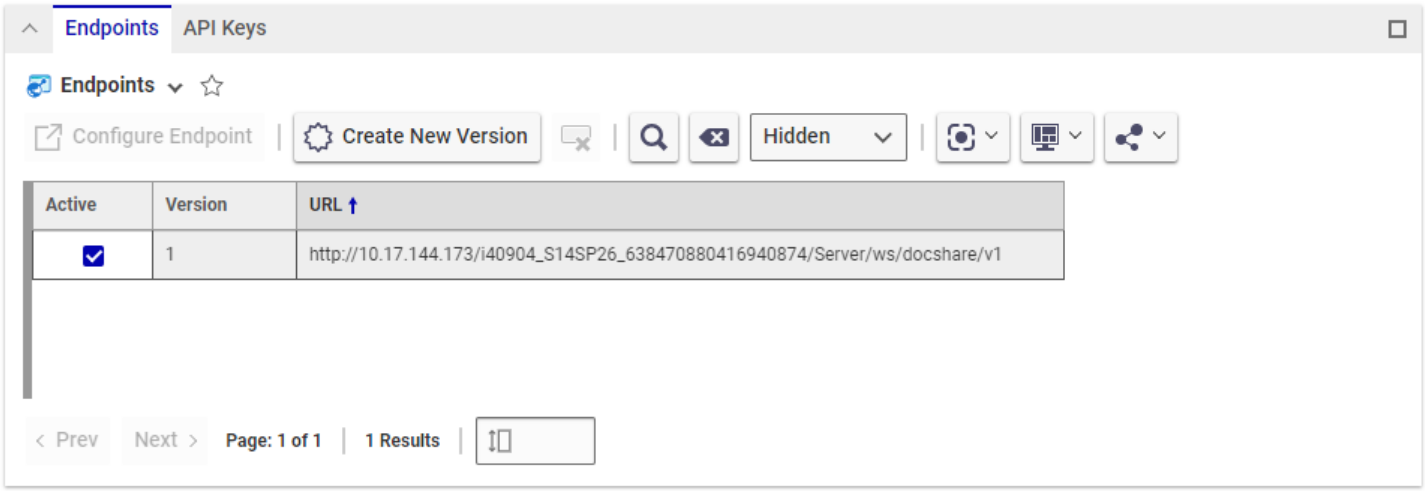
Each Aras Innovator instance has one endpoint for the default REST API. This default endpoint URL uses the following format:

```
{Innovator_url}/Server/odata/{…}
```

However, CWS can produce multiple REST APIs – each with a unique endpoint URL in the following format:

`{Innovator_url}/Server/ws/{endpoint_name}/v{endpoint_version}/{…}`

To confirm the Endpoint URL(s) for a REST API published via CWS, open the **Web Service item** and check the **Endpoints** tab.



Request Authorization

All requests sent to the Aras Innovator server must be properly authorized. For the platform's default REST API, that means an application must request an access token from the OAuth server and include it in the headers of each HTTP request. CWS supports both OAuth access tokens and API Keys. The following sections outline how to use each authorization approach in a request to a CWS Endpoint.

OAuth Token

The process for using an OAuth access token with a CWS Endpoint is the same as the process for the default REST API.

> **Important**
>
> Refer to section **Using OAuth 2.0 Tokens from the Authentication Server** in the Aras Innovator REST API documentation for step-by-step instructions.

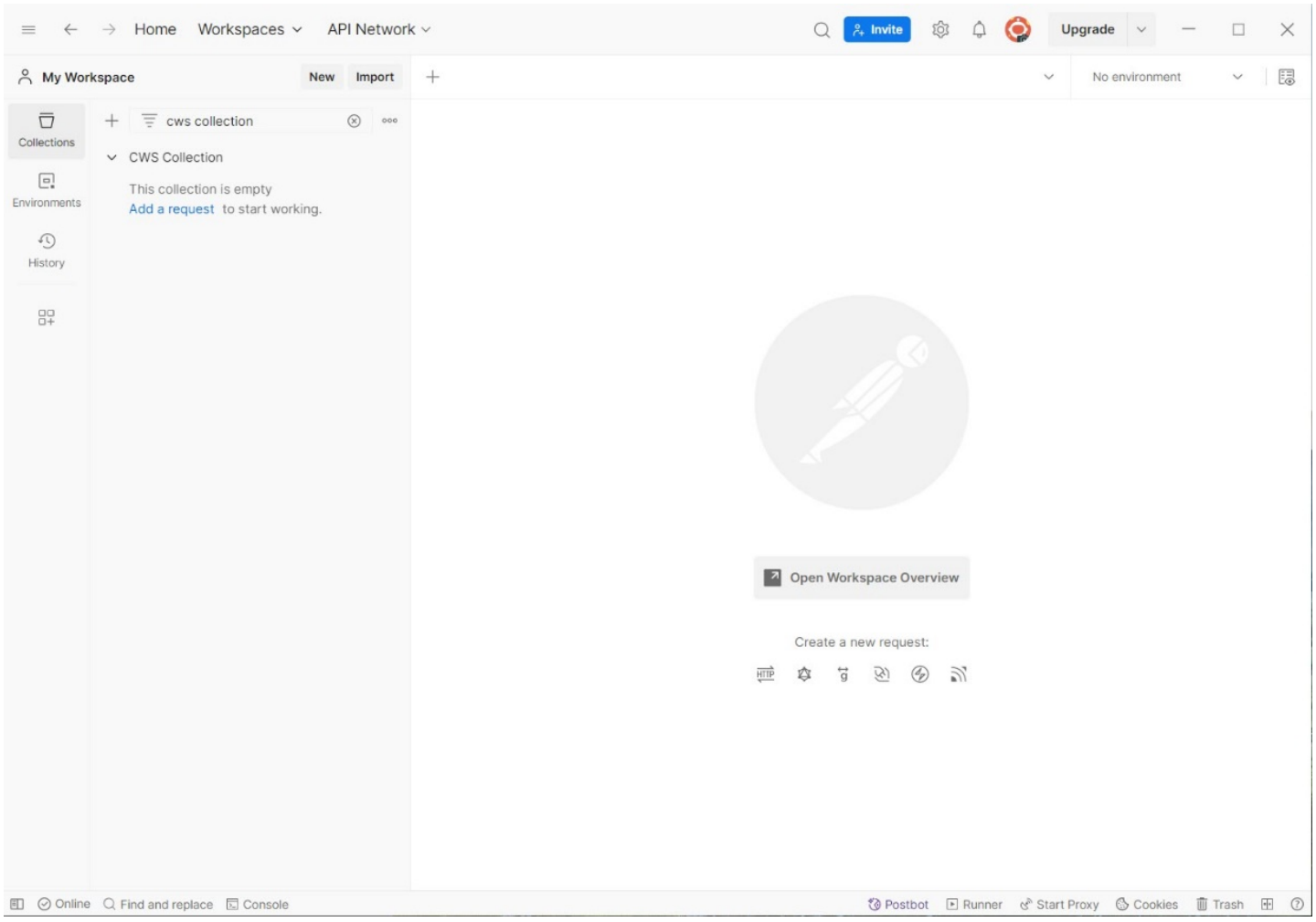
API Key

The following steps outline the process of using an API Key to authorize a request to a CWS Endpoint.

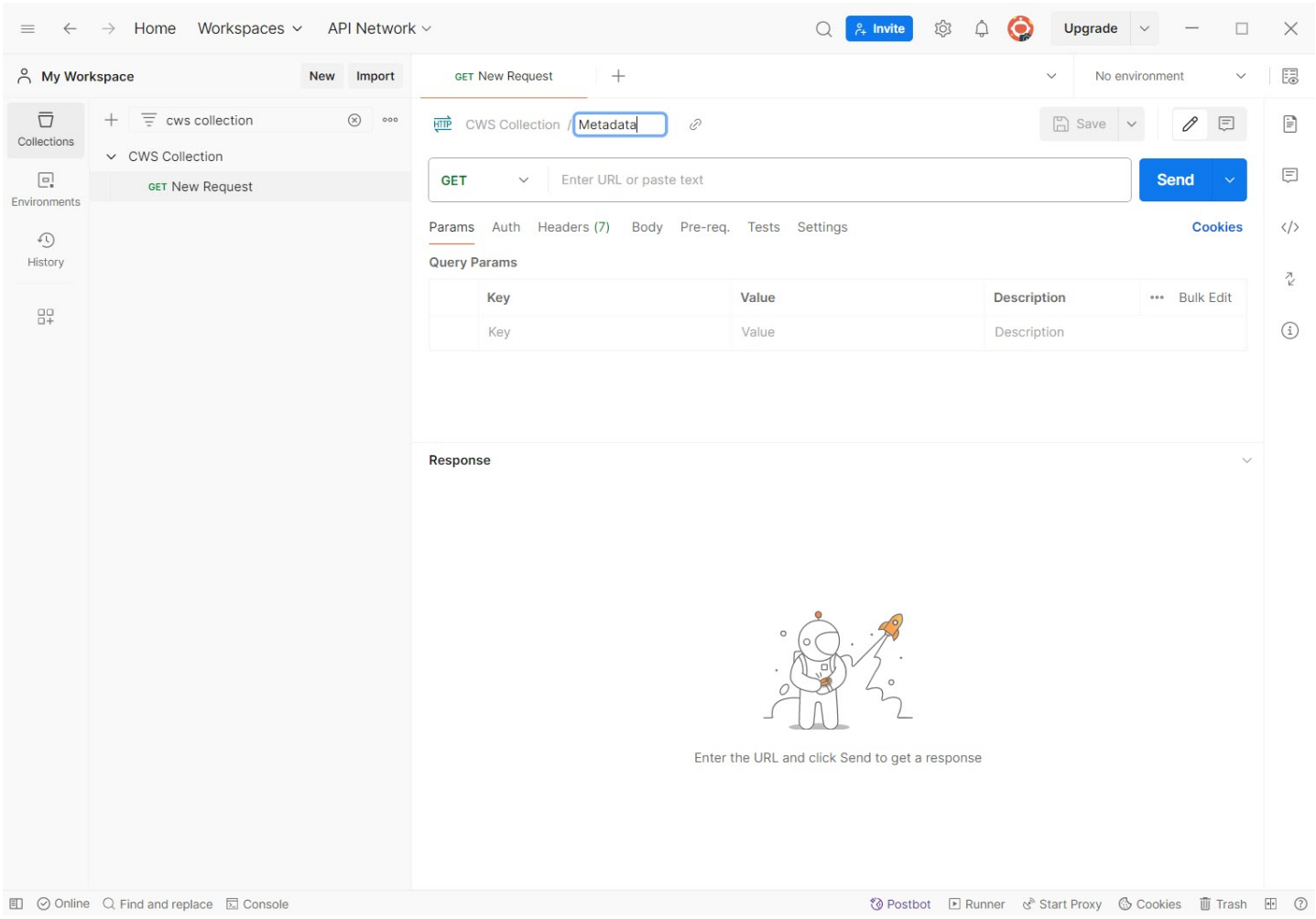
Important

This example uses the Postman HTTP client on the desktop. However, the general concept of passing the API Key in the request's Authorization header applies to any HTTP request.

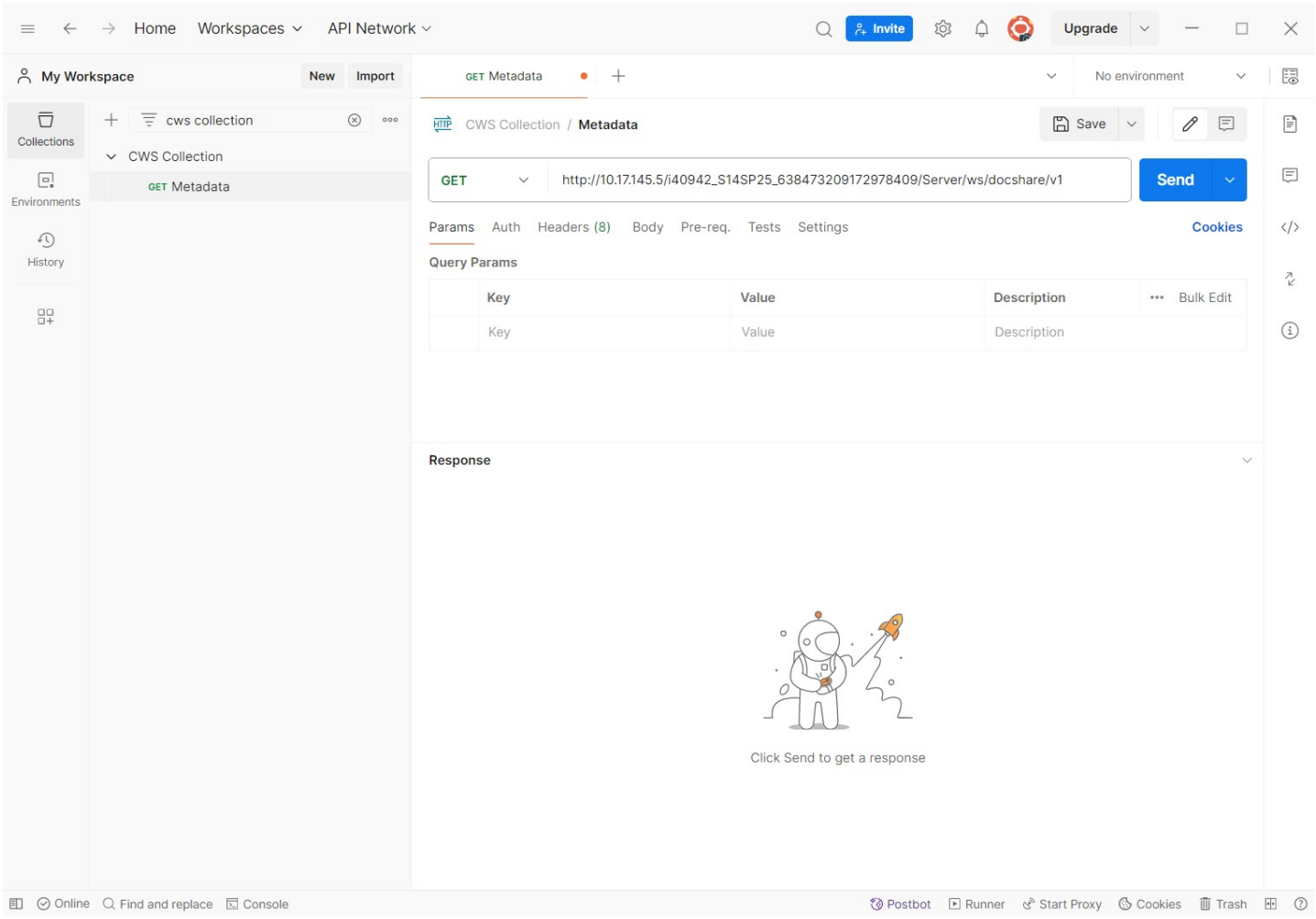
1. Download the Postman application on the local machine.
2. Under **My Workspace**, select **Collections** and click **Create New Collection**.
3. Provide a **name** for the new Workspace. For example, "CWS Collection".



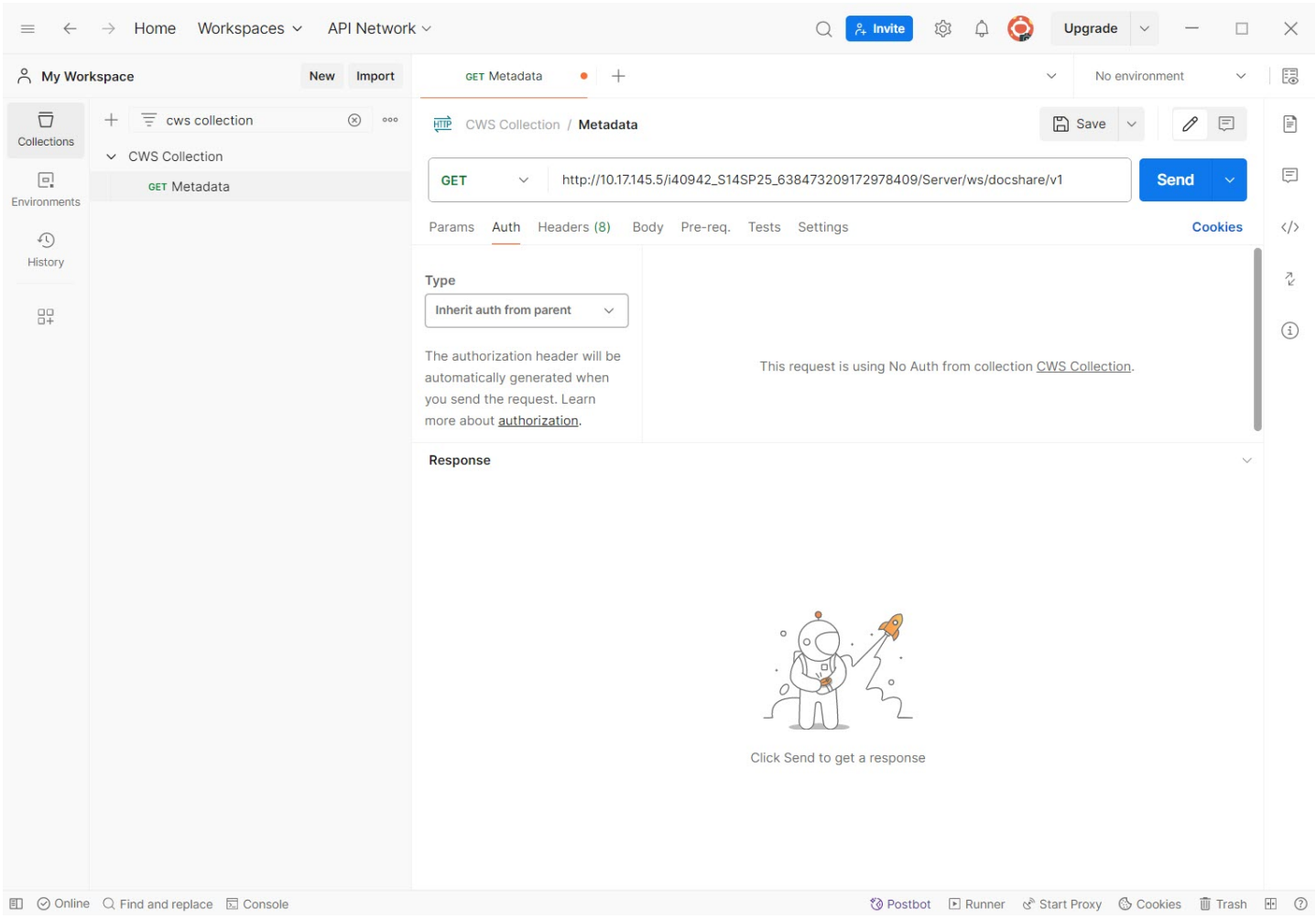
4. Click **Add a request** under the new Collection.



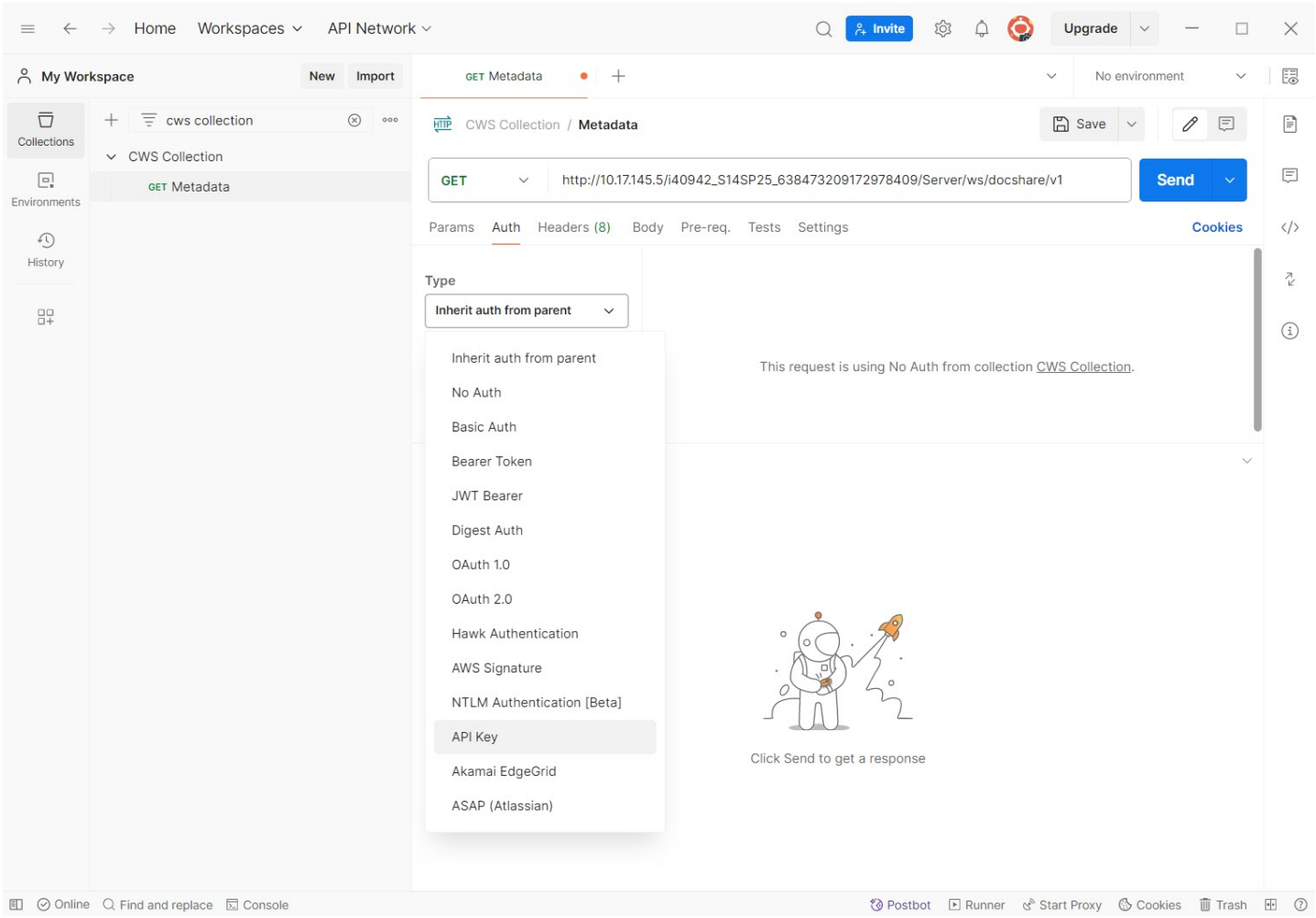
5. Provide a **name** for the new Request. For example, “Metadata”.
6. In the address bar, enter the CWS **Endpoint URL**.



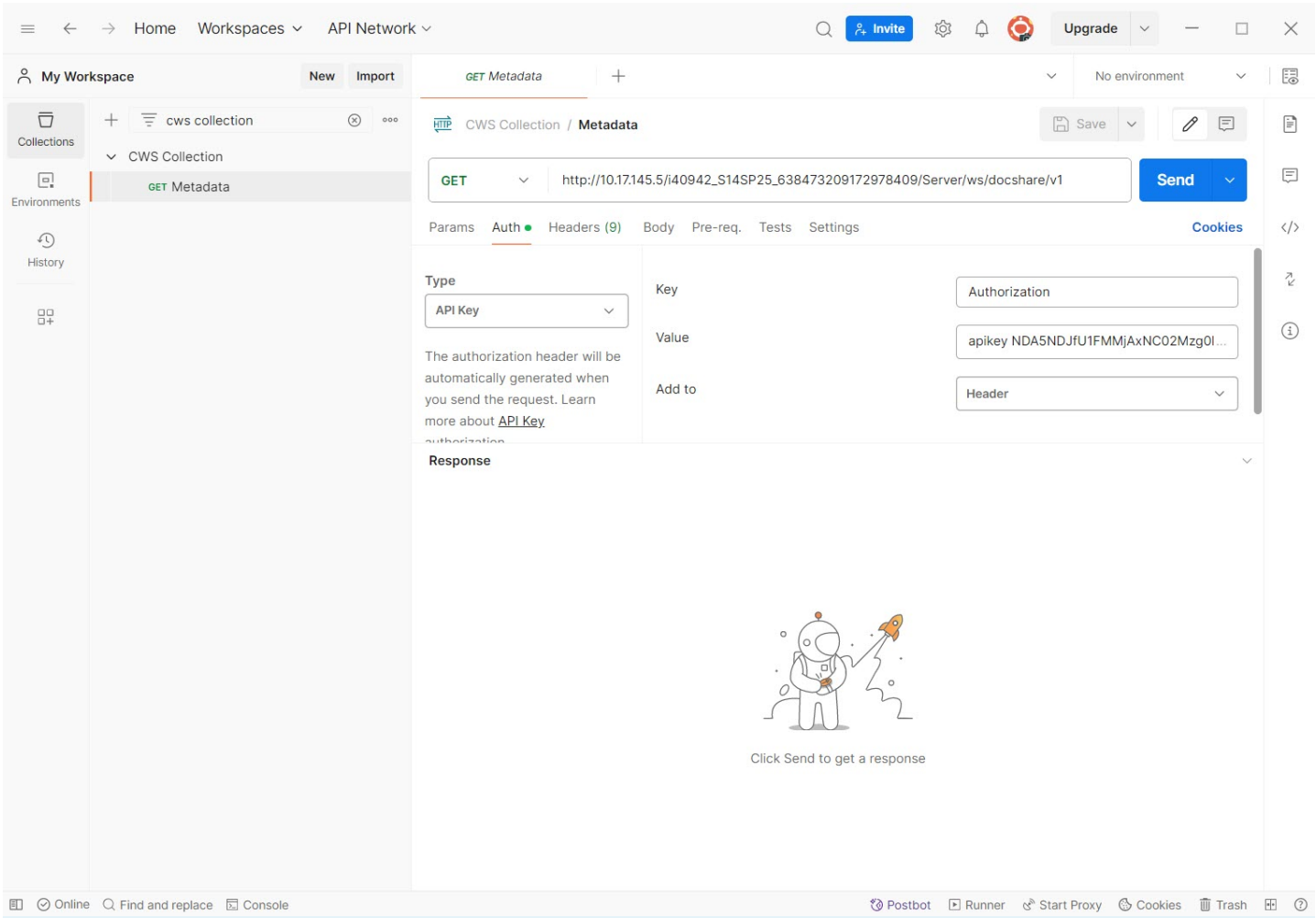
7. Click the **Auth** tab.



8. Select **API Key** from the **Type** dropdown.



9. In the Value field, enter **apikey** followed by an API Key generated with the steps in section **Creating an API Key**.

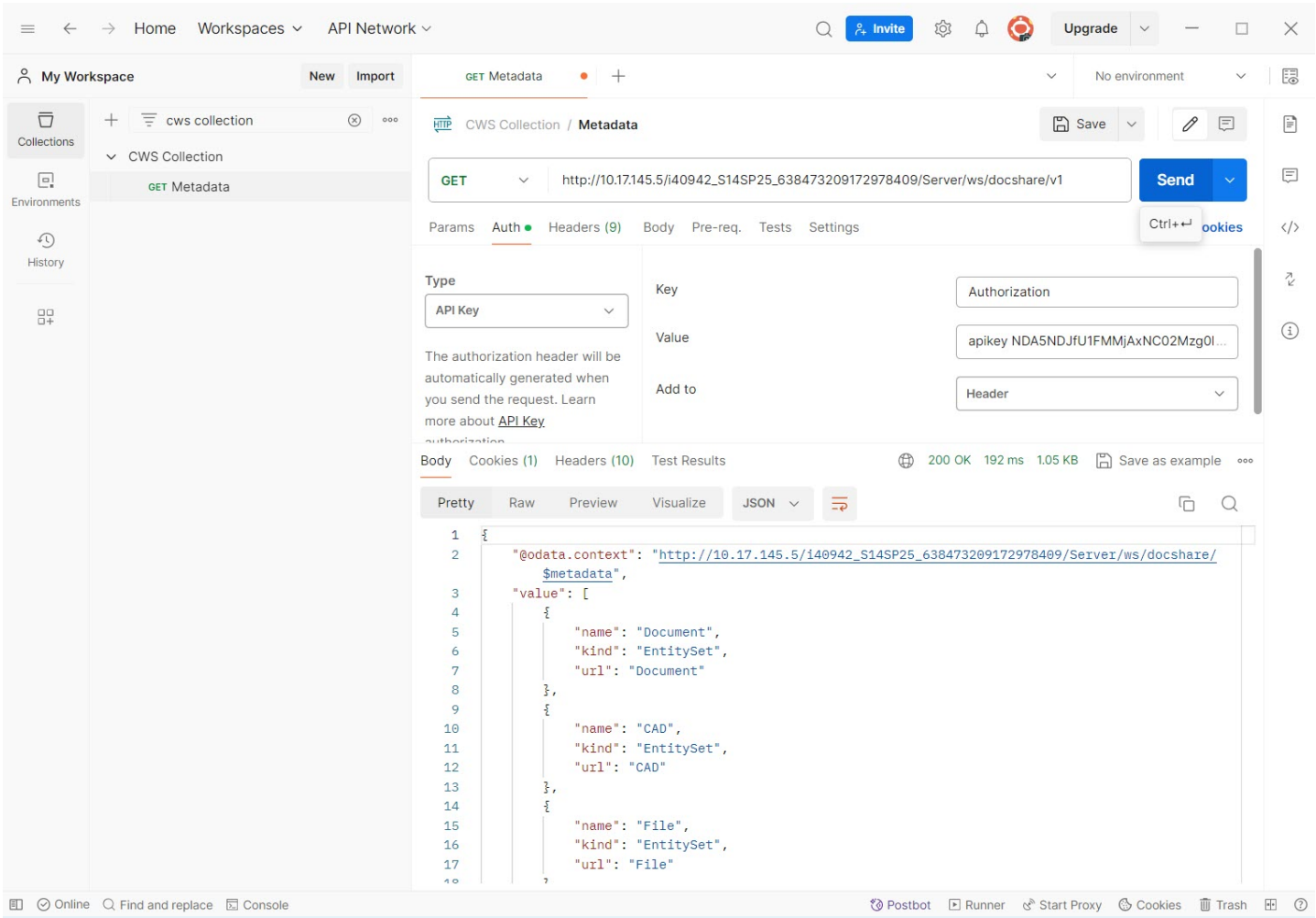


Important

It's important to include a space between the "apikey" prefix and the key's value.

10. Click **Send**.

    The request should return a 200 OK response and the response body will appear in the bottom pane of the Postman application.

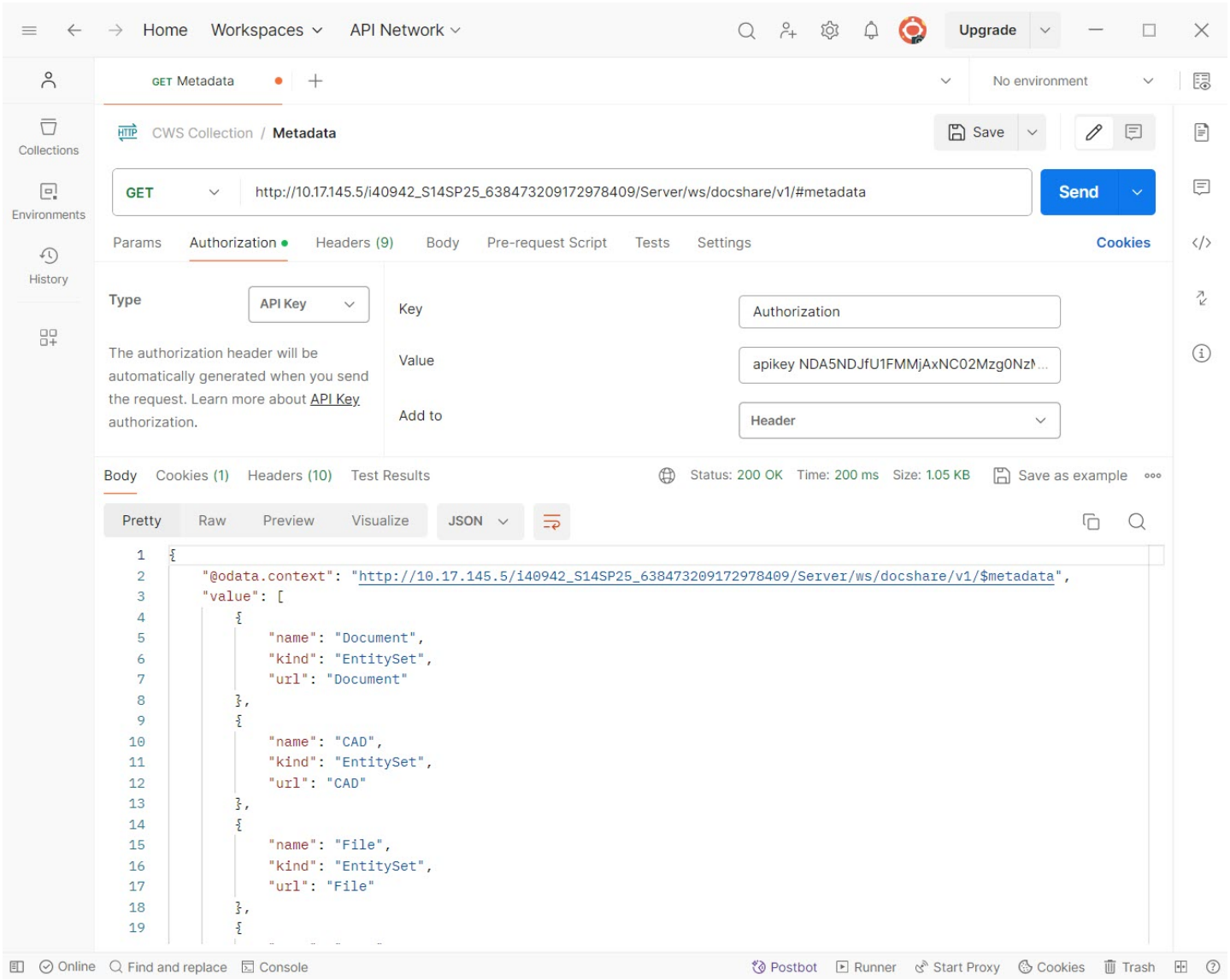


Important

The Authorization header is the default location for passing an API key in a CWS request. However, the server will check other non-reserved headers for a valid API key with the prefix "apikey_" if the Authorization header of a request is not populated.

Metadata Endpoint

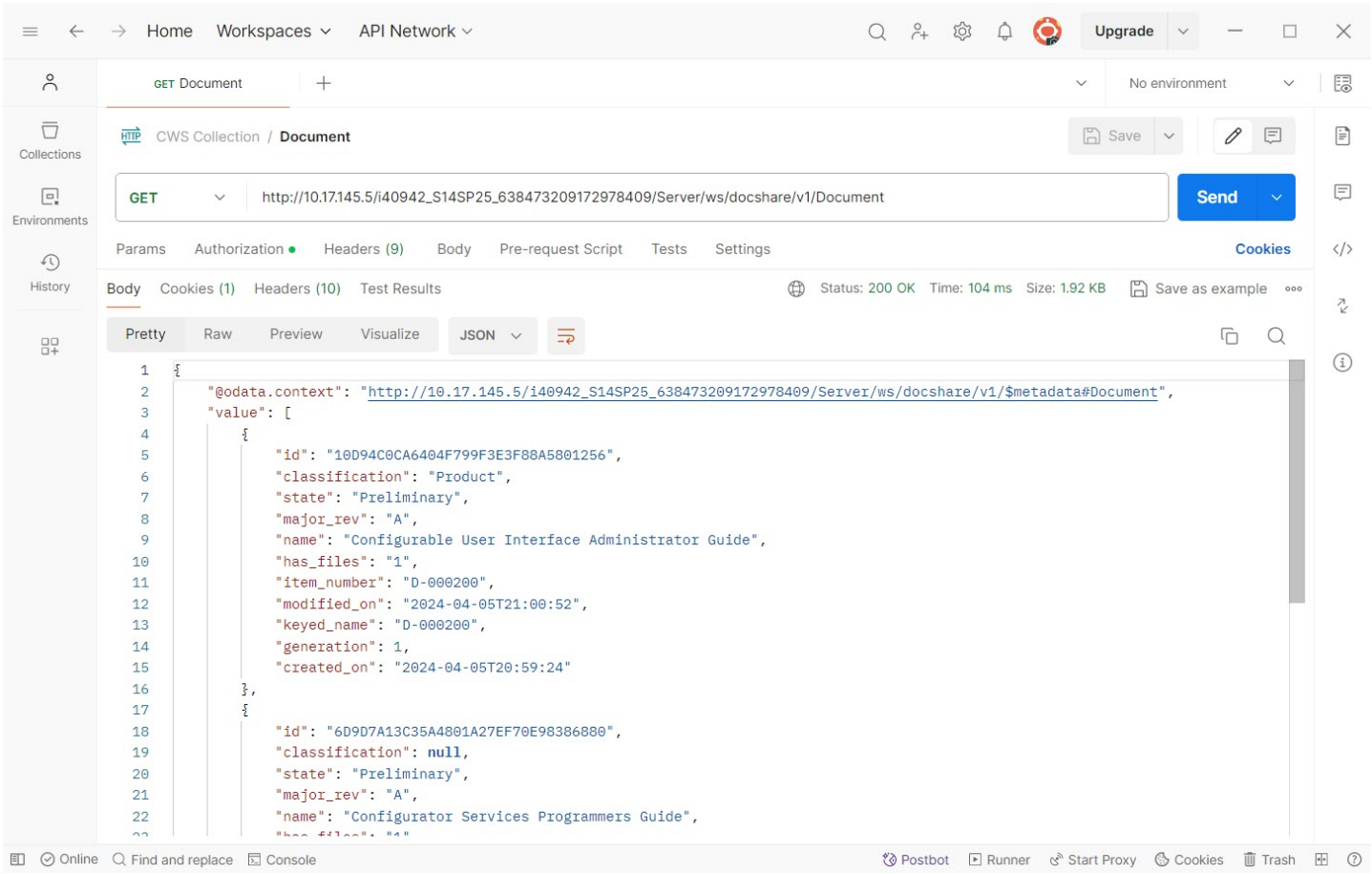
REST APIs published with CWS have a special metadata endpoint that describes the API's schema in an Odata-compliant format. This schema can be accessed with the Endpoint url as shown in the previous section. It can also be accessed by appending #metadata to the Endpoint url.



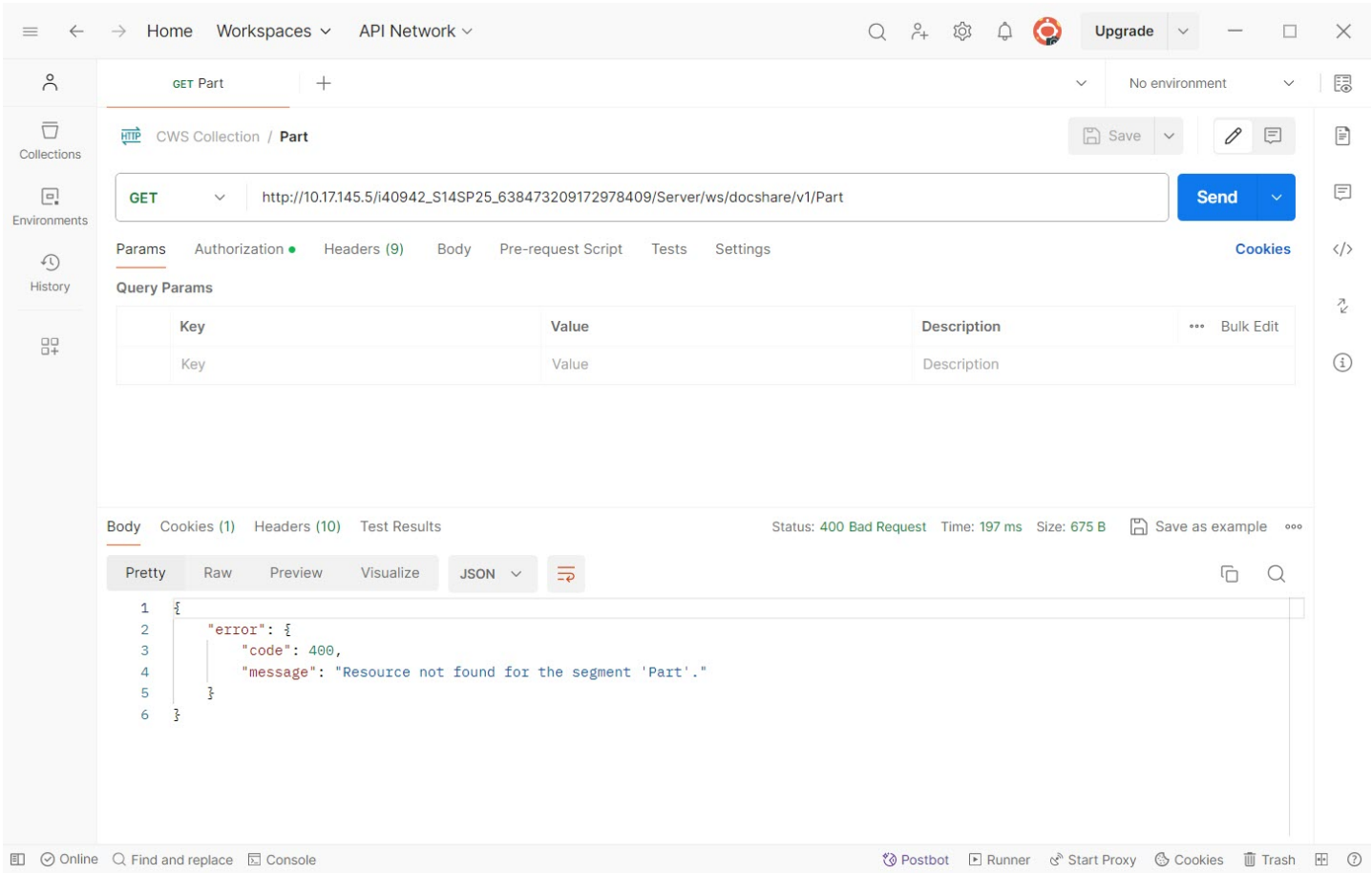
Entities

The scope of the platform's default REST API includes all ItemTypes and server Methods in the Aras Innovator database. This contrasts with the scope of CWS-published APIs, which include only the data and logic described in the Endpoint configuration.

For example, CWS Endpoint that permits "get all" for Documents will respond with a 200 OK status and Document data in the response body.



However, the same CWS Endpoint will respond with a 400 Bad Request status if it receives a request for an ItemType that is not included in the scope.



Batch Requests

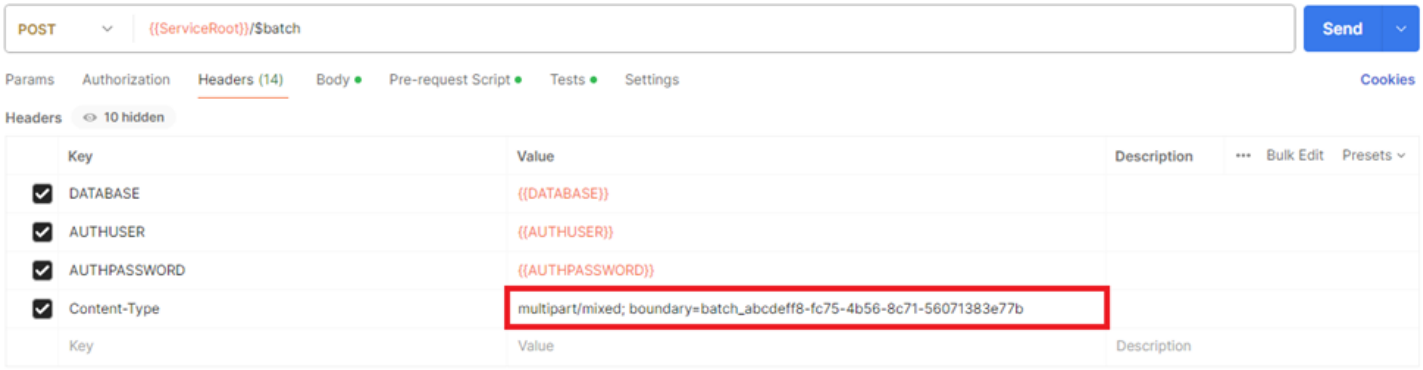
In OData, batching enables developers to bundle multiple requests into a single call to the *$batch* endpoint (see OData documentation). A batch request is represented as a Multipart MIME message, a standard format allowing the representation of multiple parts, each of which may have a different content type, within a single request.

Batch requests are submitted as a single **HTTP POST** request to the $batch endpoint of a Web Service. Any requests nested in a batch are executed sequentially and synchronously.

Batch Request Headers

This section describes the headers for an HTTP POST request to the $batch endpoint.

- **Content-Type**: Required. This header must have the multipart/mixed value and a boundary specification as defined in the Batch Request Body section (see an example below).



- **OData-Version**: Optional. If using this header, set the value to *4.0*.
- **Prefer**: Optional. By default, the CWS engine stops processing a batch request when any of the nested requests return an error. To continue processing the batch in case of an error in a nested request, include the Prefer header with the value `odata.continue-on-error` .

Batch Request Body

The body of a batch request comprises a series of individual requests and Change Sets , each represented as a distinct MIME part (i.e. separated by the boundary defined in the *Content-Type* header).

An individual request in the context of a batch request can be

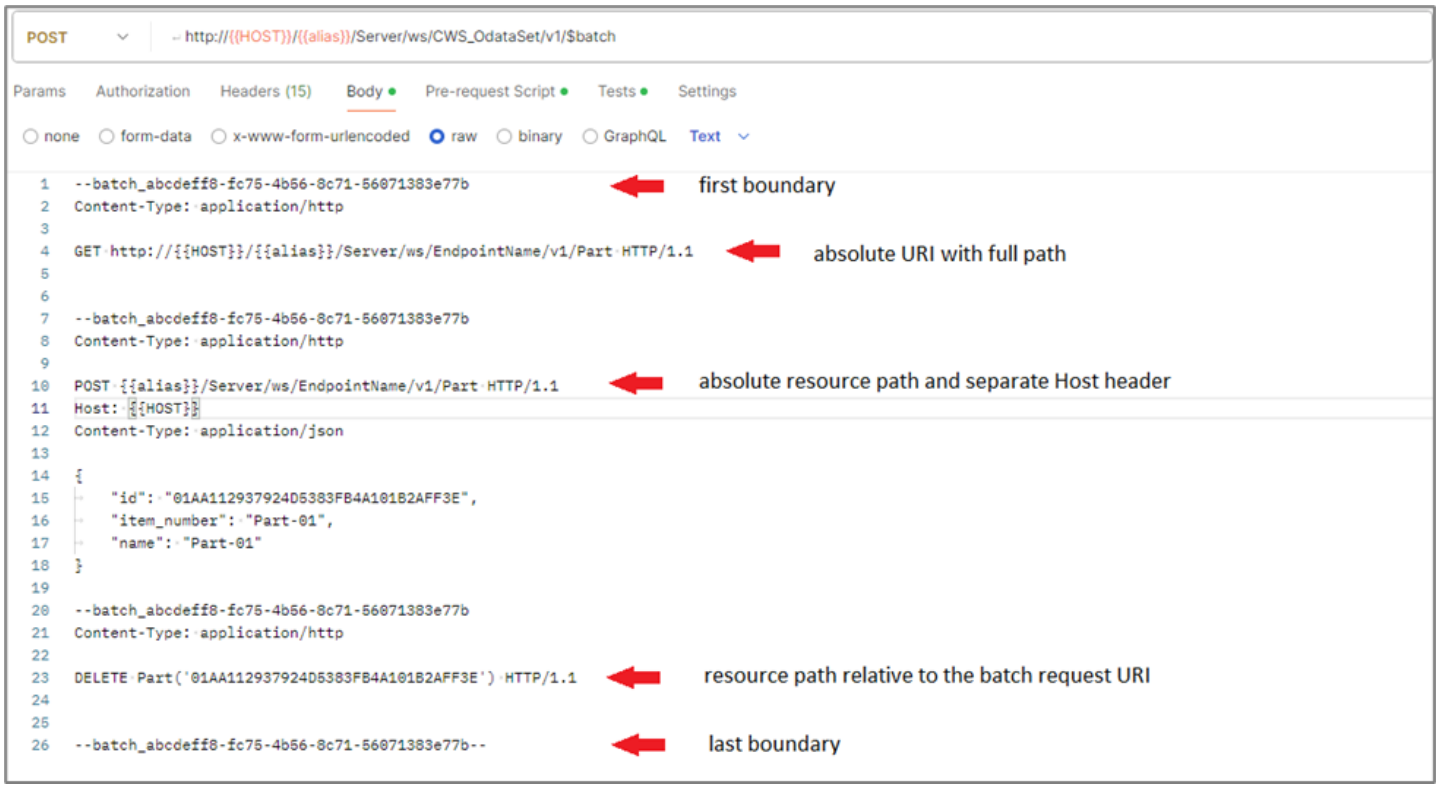
- Data request ;
- Data Modification request;
- Action invocation request.

A MIME part representing an individual request must include a *Content-Type* header with the value *application/http* and should contain a *Content-Transfer-Encoding* header with the value *binary*.

CWS supports three Request-URI formats of HTTP requests serialized within MIME part bodies:

- Absolute URI with schema, host, port, and absolute resource path
    - The absolute URI of each request in a batch must be the same as the URI of the source batch request.
- Absolute resource path and separate Host header
- Resource path relative to the batch request URI

The following sample batch request demonstrates each of these formats.



Important

The formatting of the batch request body is important. Each part must be separated by an empty line. Add an extra empty line after requests without a body (see line #4 above).

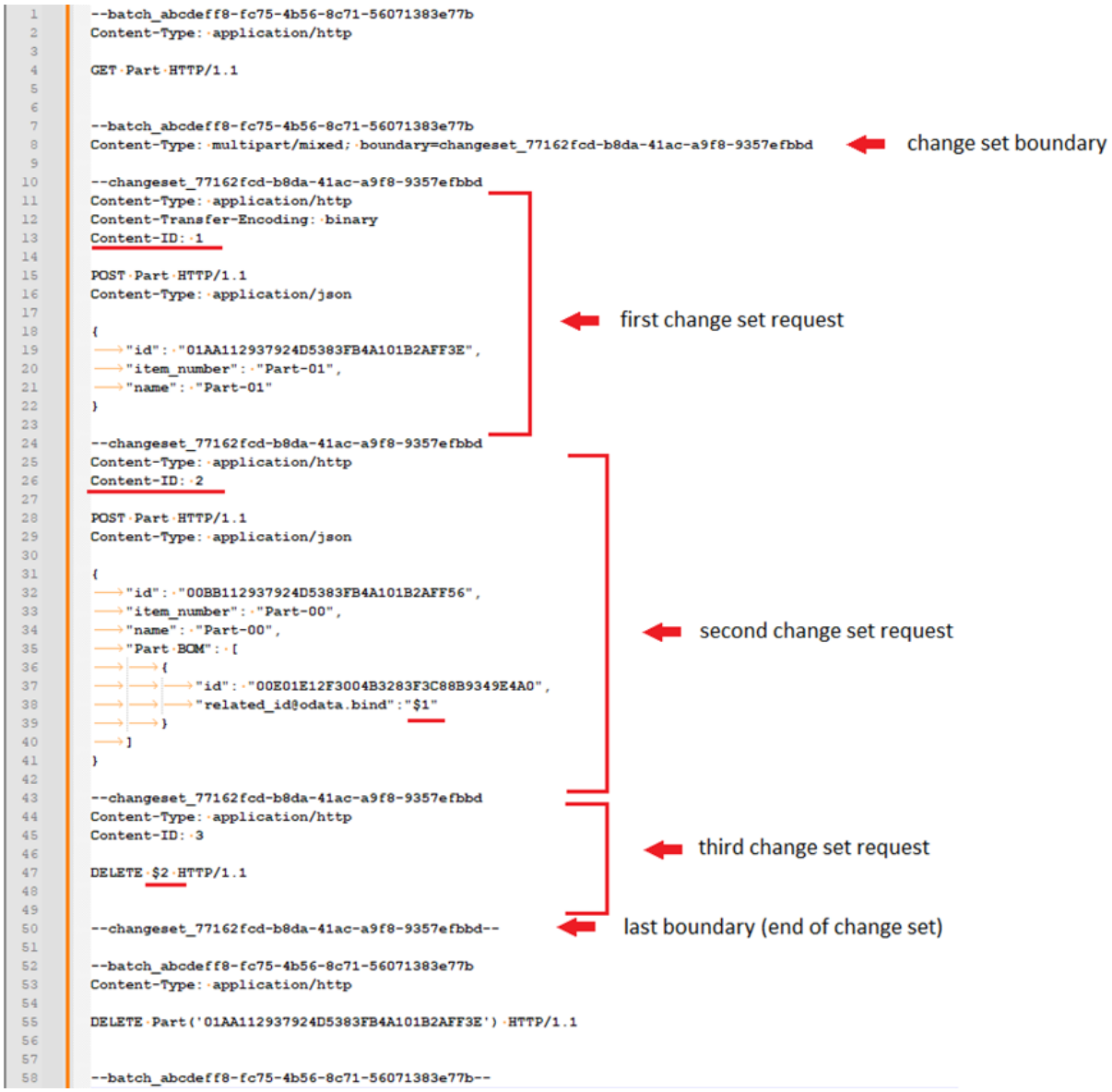
Change Sets

A **Change Set** is an atomic unit of work consisting of an unordered group of one or more Data Modification requests or Action invocation requests.

All Change Sets must meet the following requirements:

- Change Sets may not contain any GET requests or other Change Sets.
- The order of requests within a Change Set is significant; the requests within a Change Set are processed sequentially.
- If one request within a Change Set fails, then all requests in the Change Set are rolled back.

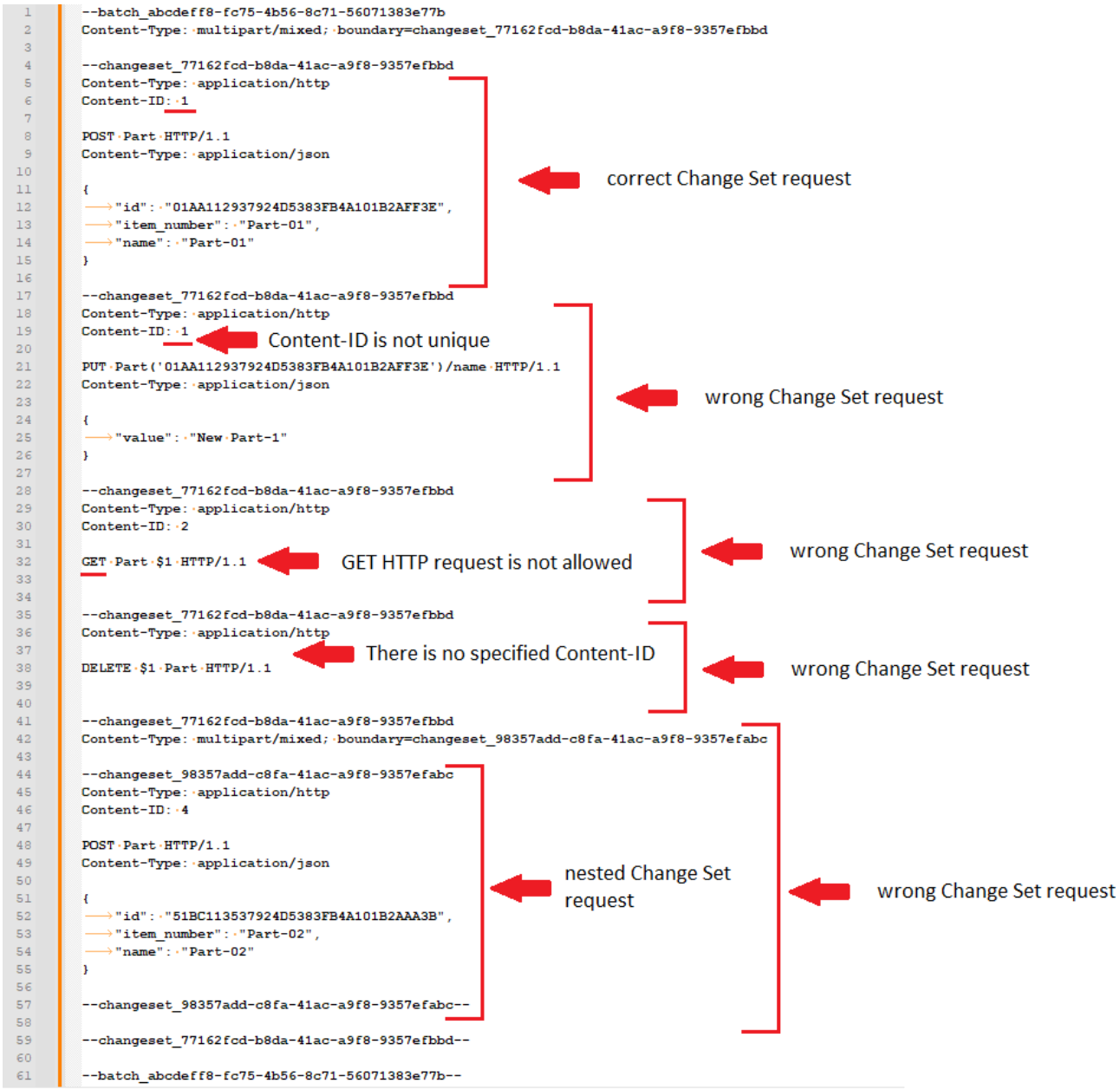
- Each Change Set must be a multipart MIME document with one part for each operation that makes up the Change Set.
- Each part representing an operation in the Change Set must include the same headers (Content-Type and Content-Transfer-Encoding) and associated values as previously described for operations (see below).




```
--batch_abcdeff8-fc75-4b56-8c71-56071383e77b
Content-Type: application/http

GET Part HTTP/1.1


--batch_abcdeff8-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-Transfer-Encoding: binary
Content-ID: 1

POST Part HTTP/1.1
Content-Type: application/json

{
    "id": "01AA112937924D5383FB4A101B2AFF3E",
    "item_number": "Part-01",
    "name": "Part-01"
}

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 2

POST Part HTTP/1.1
Content-Type: application/json

{
    "id": "00BB112937924D5383FB4A101B2AFF56",
    "item_number": "Part-00",
    "name": "Part-00",
    "Part BOM": [
        {
            "id": "00E01E12F3004B3283F3C88B9349E4A0",
            "related_id@odata.bind":"$1"
        }
    ]
}

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 3

DELETE $2 HTTP/1.1


--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--

--batch_abcdeff8-fc75-4b56-8c71-56071383e77b
Content-Type: application/http

DELETE Part('01AA112937924D5383FB4A101B2AFF3E') HTTP/1.1


--batch_abcdeff8-fc75-4b56-8c71-56071383e77b--
```

- Each request within a Change Set must specify a *Content-ID* header with a value unique within the batch request.

- Entities created by an request within a Change Set can be referenced by subsequent requests within the same Change Set in places where a resource path to an existing entity can be specified. The temporary resource path for a newly inserted entity is the value of the Content-ID header prefixed with a $ character. Examples of correct and wrong Change Set requests are below.



Responding to a Batch Request

Requests within a batch are evaluated according to the same semantics used when processing individual requests. The following response was generated by the sample batch request above.

```
--batch_abcdeff8-fc75-4b56-8c71-56071383e77b
Content-Type: application/http

HTTP/1.1 200 OK
OData-Version:4.0
Content-Type:application/json;odata.metadata=minimal;IEEE754Compatible=false

{
  "@odata.context": "http://{{host}}/{{alias}}/Server/ws/EndpointName/v1/$metadata#Part",
  "value": []
}
--batch_abcdeff8-fc75-4b56-8c71-56071383e77b
Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd

--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 1

HTTP/1.1 201 Created
Preference-Applied:return=representation
Location:http://{{host}}/{{alias}}/Server/ws/EndpointName/v1/Part('01AA112937924D5383FB4A101B2AFF3E')
OData-Version:4.0
Content-Type:application/json;odata.metadata=minimal;IEEE754Compatible=false

{
  "@odata.context": "http://{{host}}/{{alias}}/Server/ws/EndpointName/v1/$metadata#Part/$entity",
  "id": "01AA112937924D5383FB4A101B2AFF3E",
  "major_rev": "A",
```

```
  "name": "Part-01",
  "item_number": "Part-01"
}
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 2

HTTP/1.1 201 Created
Preference-Applied:return=representation
Location:http://{{host}}/{{alias}}/Server/ws/EndpointName/v1/Part('00BB112937924D5383FB4A101B2AFF56')
OData-Version:4.0
Content-Type:application/json;odata.metadata=minimal;IEEE754Compatible=false

{
  "@odata.context": "http://{{host}}/{{alias}}/Server/ws/EndpointName/v1/$metadata#Part/$entity",
  "id": "00BB112937924D5383FB4A101B2AFF56",
  "major_rev": "A",
  "name": "Part-00",
  "item_number": "Part-00",
  "Part_BOM": [
    {
      "id": "00E01E12F3004B3283F3C88B9349E4A0",
      "sort_order": 128
    }
  ]
}
--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
Content-Type: application/http
Content-ID: 3
```

```
HTTP/1.1 204 NoContent
OData-Version:4.0
Content-Type:text/plain



--changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
--batch_abcdeff8-fc75-4b56-8c71-56071383e77b
Content-Type: application/http

HTTP/1.1 204 NoContent
OData-Version:4.0
Content-Type:text/plain



--batch_abcdeff8-fc75-4b56-8c71-56071383e77b--
```

Overwriting System Properties

CWS provides the ability to overwrite the value of system properties in the Aras Innovator database. This is unique to CWS, as it is not possible to overwrite system properties with AML, the IOM, or the default OData REST API.

Override Criteria

This capability requires 3 criteria to be met to overwrite system properties with CWS:

1. The HTTP request must use the POST, PUT, or PATCH action
2. The body of the HTTP request must contain one or more system properties included in the CWS endpoint's schema.

   Example:

    - created_by_id
    - created_on
    - current_state
    - generation
    - is_released
    - modified_on
    - modified_by_id
    - state
    - not_lockable
3. The HTTP request is authorized by an API Key with the “Override System Properties” scope.

Important

Any request that meets the criteria above will overwrite system property data in the Aras Innovator database. Only use this capability when a use case requires an application or integration to overwrite the system property values in the Aras Innovator database. This feature may replace critical metadata such as created_by_id, modified_by_id, created_on, modified_on, state, etc.

Requirements and Considerations

The following system properties have requirements or conditions to consider when overwriting their values.

- **state**: Contains the string label of the item’s current Lifecycle State. Overwriting this property without also setting **current_state** will not change the label shown in the Aras Innovator client.
- **current_state**: Contains the id of the item’s Lifecycle State. Overwriting this property automatically sets the **state** property.
- **generation**: Value must form a unique pair with the item’s **config_id**. The database may not contain two items with the same **config_id** and **generation**.
- **is_current**: Adding a new version of an item does not automatically update the **is_current** property of the item’s previous current generation. The database may not contain two items with the same **config_id** and **is_current**=1.
- **is_released**: Items with **is_released**=1 should also have **not_lockable**=1. This needs to be set explicitly and is not synced automatically.
- **created_by_id**: Changing this property will affect any permissions for this item referencing the Creator identity.
- **current_state**: Items may inherit permissions from their current Lifecycle State. Changing this property may affect the permissions of the current item.

Important

Overwriting system properties may create inconsistent data in the Aras Innovator database and should be done carefully. The HTTP response from the server is not guaranteed to throw errors caused by inconsistent data.

Server Events

This capability leverages the Data Sync Service (DSS) engine to handle requests that set and update system properties. As a result, these requests will not trigger standard ItemType server events like onAfterAdd, onBeforeUpdate, etc. Instead, the DSS engine will execute onAfterSyncAdd, onBeforeSyncUpdate, etc. This provides the ability to have custom server event logic for use cases that overwrite system properties.

Refer to the **Item Actions** section of the **Aras Innovator 40: Data Synchronization Services Programmer's Guide** for more information.

Important

If no sync server events are events configured on the ItemType, the overwrite system property requests will not trigger any server event methods.

Permissions

Though API Keys with the "Override System Properties" scope can overwrite system properties, the user associated with the API Key must have permission to modify the data in the request. This requires configuring standard permissions, Can Add rights, MAC policies, and/or DAC policies.

Sending a Request

Overwriting system properties with CWS follows the same general process as sending other requests to the CWS endpoint. The following steps outline the general process. See the referred content for more information.

1. Refer to the section **Creating an API Key** above to create an API Key with the scope **Override System Properties**.
2. Refer to the **API Key** section under **Request Authorization** to create an HTTP POST, PUT, or PATCH request authorized by the API Key with the **Override System Properties** scope.
3. Include one or more system properties in the body of the request

4. Send the request to the CWS endpoint

Uploading Files

CWS provides an upload endpoint to quickly add files to the Aras Innovator vault with a single request. This is more convenient for developers and enables integrations with systems where developers might not be able to make multiple calls to upload a file (for example, when using webhooks).

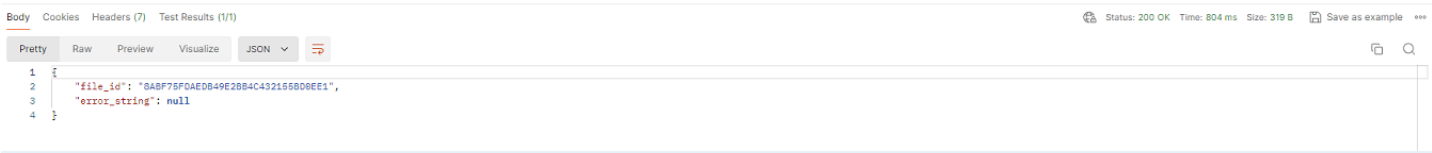
File Upload Request

The following steps outline the process of uploading a file to the vault's upload endpoint.

1. Create an **HTTPPOST** request
2. Include an API Key or OAuth 2.0 access token to **authorize** the request.

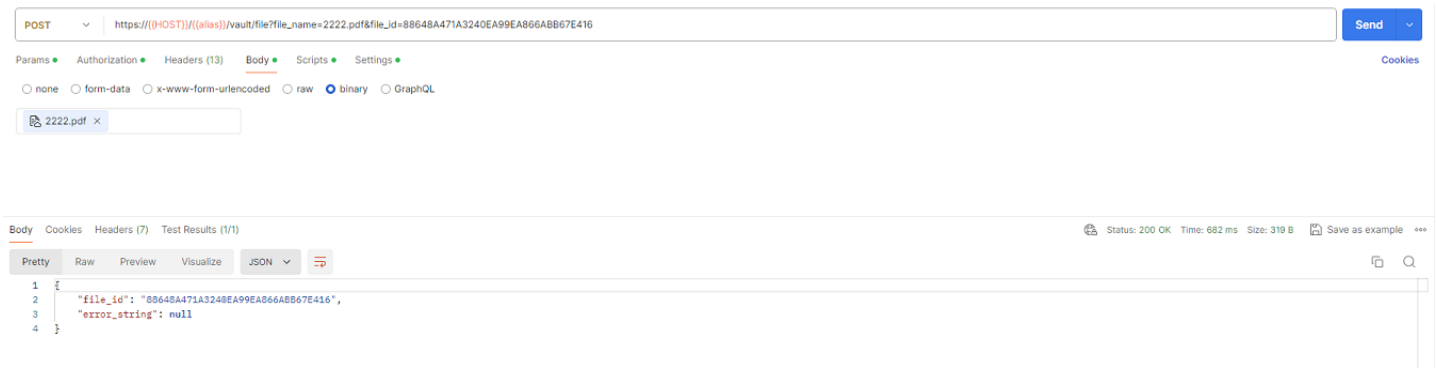
> **Important**
>
> The user associated with the API Key or access token must have permission to upload files.

3. Add the file to the **requestbody** as a binary stream.



> **Important**
>
> By default, the upload endpoint can handle a file size up to 30MB. See the Vault Configuration section below for steps to increase this limit.

4. Add the **filename** as a "file_name" query parameter.
5. Optional: Add the desired **fileid** as a "file_id" query parameter. If no id is provided, one will be generated on the server.



6. **Submit** the request.

File Upload Response

The vault will return a 200 (OK) HTTP status code and the file's id if the file is successfully uploaded. The following example shows a typical response:

```
{
    "file_id": <file ID or null>,
    "error_string": <error message or null>
}
```

Error Responses

Here are the error codes that may result from a request to the vault's upload endpoint. The error_string property in the response body also provides any error message from the vault Server.

- **413 Request Entity Too Large**: Occurs when the size of the file exceeds the limit specified in the vault configuration
- **500 Internal Server Error**: The most common causes are an invalid access token or API key, or the user does not have permission to upload files to the vault.

Vault Configuration

The following steps outline the process to increase the maximum file size accepted by the upload endpoint.

1. In the **VaultServer** folder, open **appsettings.json**.
2. Add the "MaxRequestBodySize" limit to the "Kestrel" section (add this section if needed).

3. Set the MaxRequestBodySize value (in bytes). The following example shows how to set the limit to 70MB.

```
{

    "Kestrel": {

        "Limits": {

            "MaxRequestBodySize" : 70000000

        }

    }

}
```

4. Save and close the **appsettings.json** file.
5. Open the **web.config** file in the same directory.
6. In the **system.WebServer** section, add a security section with a tag requestLimits and MaxRequestBodySize parameter.

7. Set the **MaxRequestBodySize** value (in bytes). The following example shows how to set the limit to 70MB.

```
 <?xml version="1.0" encoding="utf-8"?>

<configuration>

    <system.webServer>

        <httpProtocol>

    // httpProtocol settings

        </httpProtocol>

        <handlers>

            // handlers

        </handlers>

        <aspNetCore>

            // aspNetCore settings

        </aspNetCore>

        <security>

            <requestFiltering>

                <requestLimits maxAllowedContentLength="70000000" />

            </requestFiltering>

        </security>

    </system.webServer>
```

```
</configuration>
```

8. Save and close the **web.config** file.

Call a Server Method

Calling a server method via a CWS endpoint is similar to calling a method via the standard REST API. The method is treated as an OData action that can be called with a POST request. However, CWS also provides the ability to bind a method to a specific ItemType or restrict the data that may be passed to an “unbound” global method. The following examples demonstrate how to call methods configured with these restrictions.

> **Important**
>
> All server methods called via a CWS endpoint will return a string wrapped in a JSON object with OData metadata. For examples, see each subsection below.

ItemType Method

A web service item’s ItemType Methods are bound to a single ItemType. The server method execution context will include the id of the item the method is called on. The following example demonstrates how to call an ItemType method that returns a comma-separated list of file names related to a Document item.

Request URL Format

```
POST {endpoint_url}/{web_service_item}('id')/{method_alias}
```

Example Request URL

```
POST https://localhost/Innovator/Server/ws/DocService/v1/Document('57E624F1C81D40A6B1418FF84693FF28')/Fil
```

Example Server Method Code

```
Innovator inn = this.getInnovator();
String id = this.getID();

Item docFiles = inn.newItem("Document File","get");
docFiles.setProperty("source_id",id);
docFiles.setAttribute("select","related_id");
docFiles.getRelatedItem();
docFiles = docFiles.apply();

if (docFiles.isError()) {
  return inn.newResult("Document with id " + id + " has no related files.");
}

String result = "";
for(int i=0; i < docFiles.getItemCount(); i++) {
   Item fileItem = docFiles.getItemByIndex(i).getRelatedItem();
  result += fileItem.getProperty("keyed_name","");
  result += ",";
}

result = result.Remove(result.Length-1);
return inn.newResult(result);
```

Example Response

```
{
   "@odata.context": "https://localhost/Innovator/Server/ws/DocService/v1/$metadata#Edm.String",
   "value": "MP2960manifest.json,MP2960rootfs.rar,MP2960signature.rar,MP2960uImage.rar"
}
```

Global Method

Unlike ItemType methods, global methods do not “expect” a context item of a specific type. The following example demonstrates how to execute a global server method that requires no parameters or inputs.

Request URL Format

```
POST {endpoint_url}/{method_alias}
```

Example Request URL

```
POST https://localhost/Innovator/Server/ws/DocService/v1/CountActiveUsers
```

Example Server Method Code

```
Innovator inn = this.getInnovator();
Item activeUsers = this.newItem("User","get");
activeUsers.setProperty("logon_enabled","1");
activeUsers.setAttribute("select","id");
activeUsers = activeUsers.apply();

return inn.newResult(activeUsers.getItemCount().ToString());
```

Example Response

```
{
   "@odata.context": "https://localhost/Innovator/Server/ws/DocService/v1/$metadata#Edm.Int32",
   "value": "20"
}
```

Global Method with Parameters

If a global method requires input, parameters may be passed via the JSON body of the request. The following example demonstrates how to call a global server method that takes a file_id parameter and returns the vault url if the file is a .pptx file.

Request URL Format

```
POST {endpoint_url}/{method_alias}
```

Example Request URL

```
POST https://localhost/Innovator/Server/ws/DocService/v1/GetPptxUrl
```

Example Request Body

```
{
   "file_id":"22602CF88CC94932A2523585EB5FDB21"
}
```

Example Server Method Code

```
Innovator inn = this.getInnovator();
string fileId = this.getProperty("file_id","");

if (string.IsNullOrEmpty(fileId)) {
   return inn.newError("file_id is required");
}

Item fileItem = inn.getItemById("File",fileId);
if (fileItem.isError()) {
   return inn.newResult("No files found with id " + fileId);
}

// check file type
string fileName = fileItem.getProperty("filename");
string extension = fileName.Substring(fileName.Length - 5);
if (extension != ".pptx") {
   return inn.newResult("File with id " + fileId + " is not a .pptx file: " + extension);
}

string url = inn.getFileUrl(fileId,Aras.IOM.UrlType.SecurityToken);
return inn.newResult(url);
```

Example Response

```
{
   "@odata.context": "https://localhost/Innovator/Server/ws/DocService/v1/$metadata#Edm.String",
   "value": "{VaultServerUrl}?dbName={DB}&amp;fileId=22602CF88CC94932A2523585EB5FDB21&amp;fileName=MP0101
}
```