

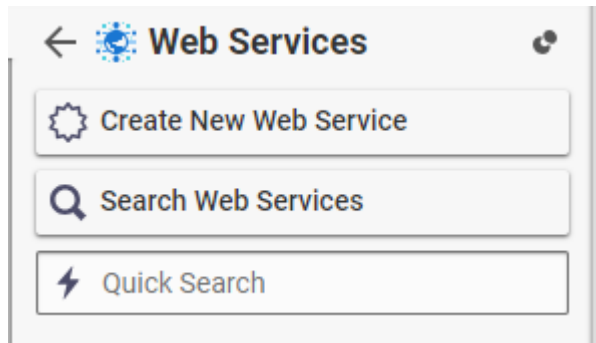
Web Service

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.

Web Service 1

Save Done Delete

Web Service

Title Endpoint Name

Description

OAuth Token Required Scope ?
Innovator

Endpoints API Keys

Endpoints ☆

Configure Endpoint Create New Version Hidden

Active	Version ↑	URL
--------	-----------	-----

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

docshare ☆

Save Done Discard

Web Service

Title Endpoint Name

Document Share docshare

Description

Share document metadata and related files with applications and integrations

OAuth Token Required Scope ?
Innovator

Endpoints API Keys

Endpoints ☆

Configure Endpoint Create New Version Hidden

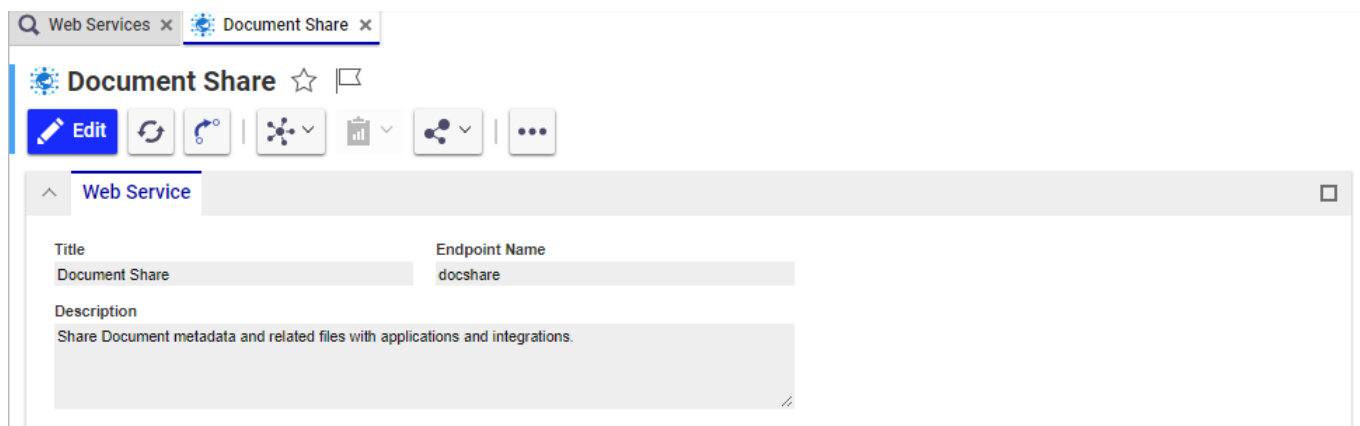
Active	Version ↑	URL
--------	-----------	-----



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:



The screenshot shows the 'Document Share' web service configuration form. At the top, there are tabs for 'Web Services' and 'Document Share'. Below the tabs is a header bar with the 'Document Share' title, a star icon, and a list icon. A toolbar contains icons for 'Edit', 'Refresh', 'Link', 'Share', 'Add', 'Remove', and 'More'. The main form area has a 'Web Service' tab selected. It contains three fields: 'Title' with the value 'Document Share', 'Endpoint Name' with the value 'docshare', and 'Description' with the text 'Share Document metadata and related files with applications and integrations.'.

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.

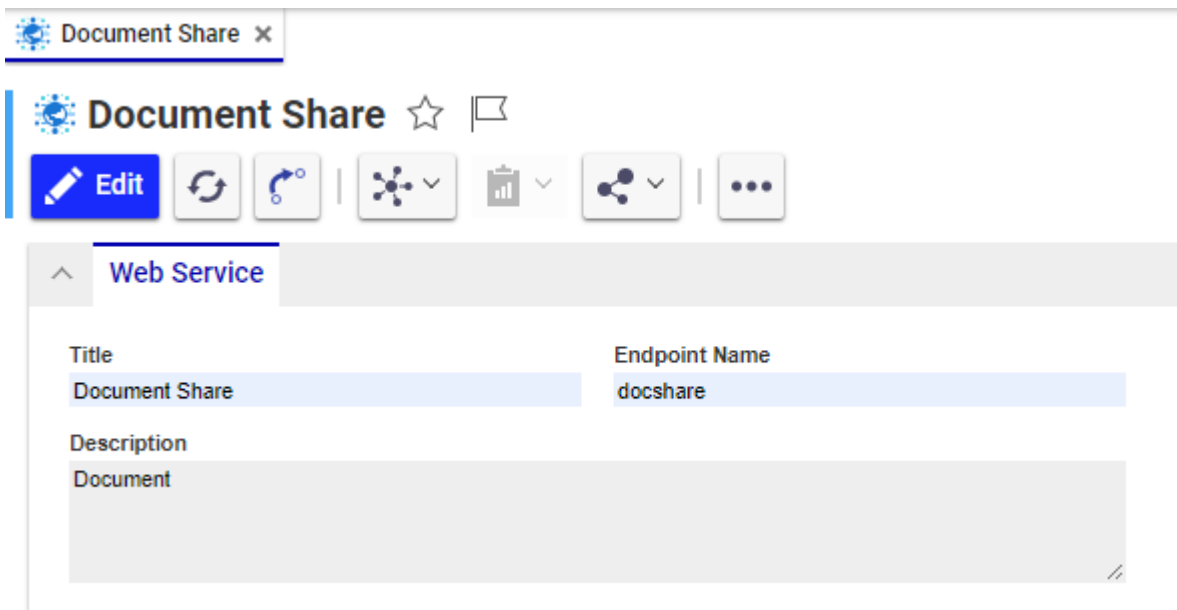
Important

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.

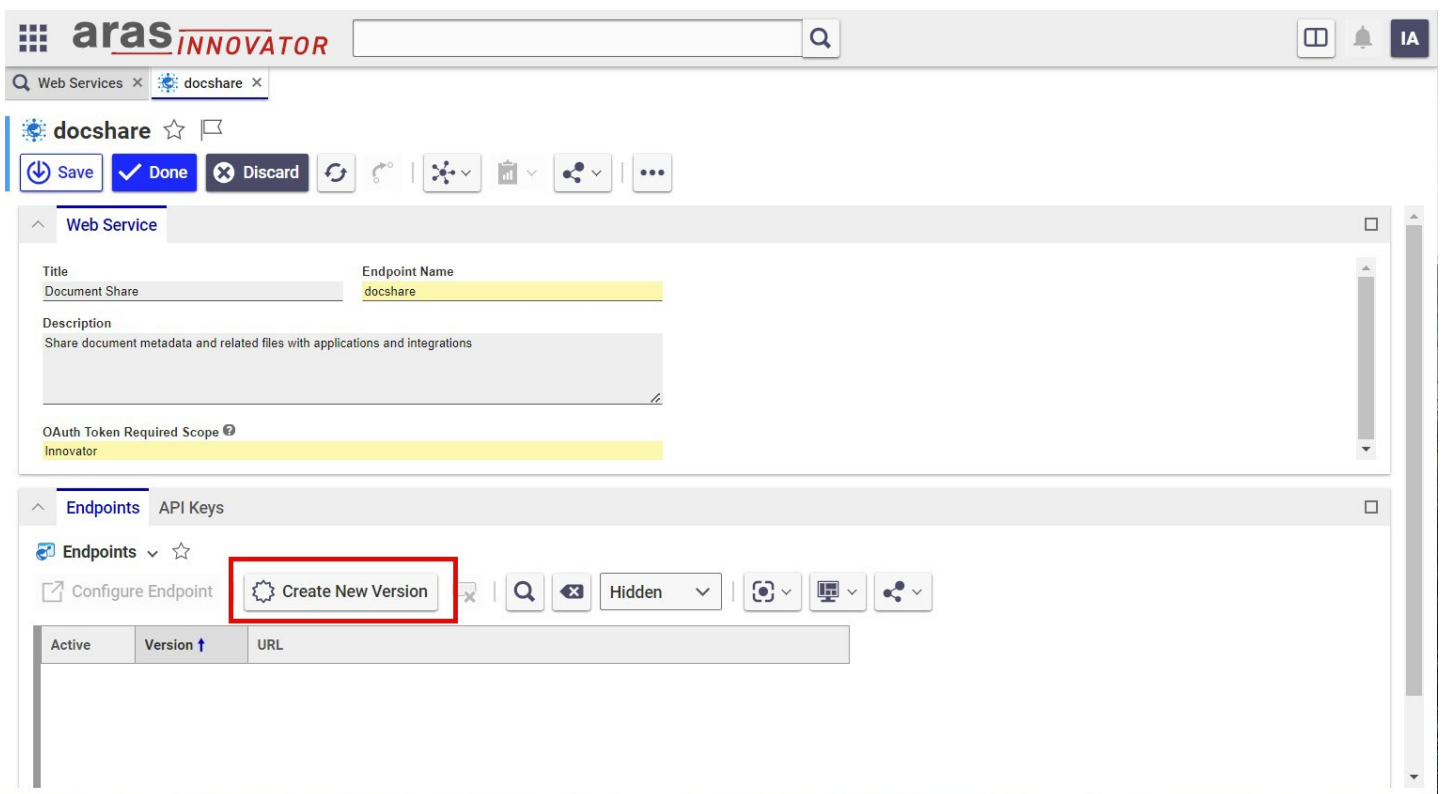
The following steps outline the process to create a new Endpoint:



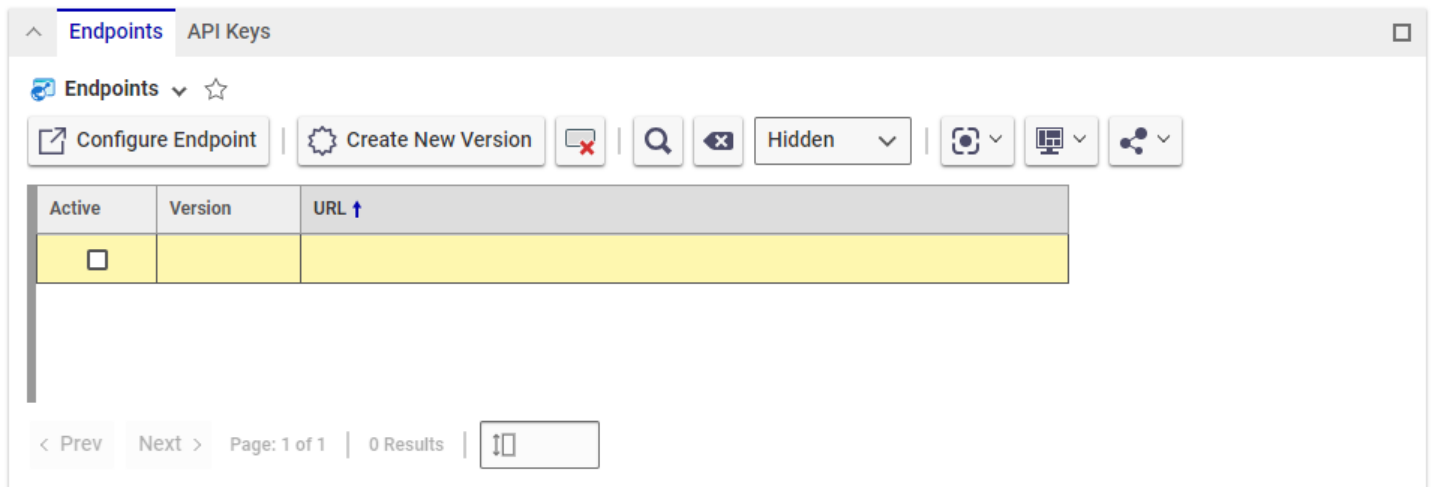
1. Open the **Web Service**.
2. Click **Edit**.



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



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



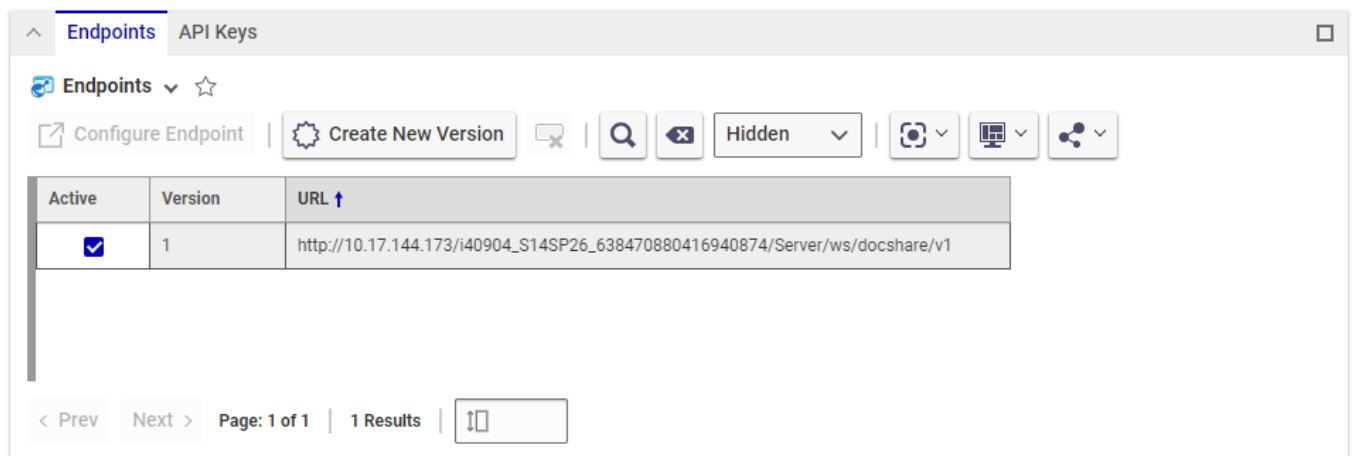
The screenshot shows the 'Endpoints' tab in a software interface. At the top, there are tabs for 'Endpoints' and 'API Keys'. Below the tabs, there is a toolbar with buttons for 'Configure Endpoint', 'Create New Version', a red 'X' icon, a magnifying glass, a left arrow, a 'Hidden' dropdown, a camera icon, a monitor icon, and a share icon. Below the toolbar is a table with three columns: 'Active', 'Version', and 'URL'. The 'Active' column has a single row with an empty checkbox. The 'Version' and 'URL' columns are empty. At the bottom, there is a pagination bar showing '< Prev', 'Next >', 'Page: 1 of 1', '0 Results', and a refresh icon.

Active	Version	URL ↑
☐		

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}`



The screenshot shows the 'Endpoints' tab in a software interface. At the top, there are tabs for 'Endpoints' and 'API Keys'. Below the tabs, there is a toolbar with buttons for 'Configure Endpoint', 'Create New Version', a red 'X' icon, a magnifying glass, a left arrow, a 'Hidden' dropdown, a camera icon, a monitor icon, and a share icon. Below the toolbar is a table with three columns: 'Active', 'Version', and 'URL'. The 'Active' column has a single row with a checked checkbox. The 'Version' column has the value '1'. The 'URL' column has the value 'http://10.17.144.173/i40904_S14SP26_638470880416940874/Server/ws/docshare/v1'. At the bottom, there is a pagination bar showing '< Prev', 'Next >', 'Page: 1 of 1', '1 Results', and a refresh icon.

Active	Version	URL ↑
☒	1	http://10.17.144.173/i40904_S14SP26_638470880416940874/Server/ws/docshare/v1

6. Select the **Endpoint**.

7. Click **Configure Endpoint**.



Endpoints API Keys

Endpoints

Configure Endpoint Create New Version Hidden

Active	Version ↑	URL
☒	1	http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1

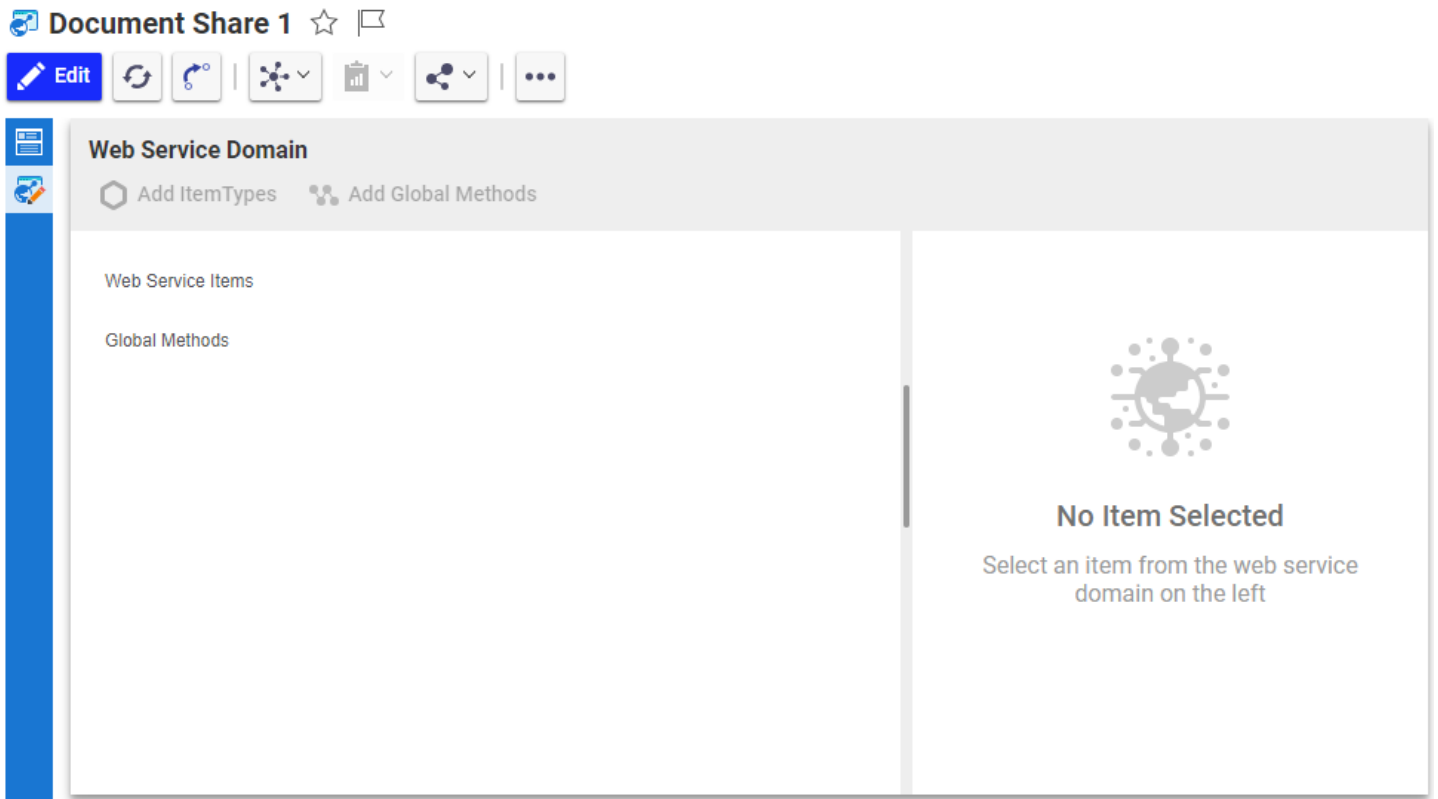
< Prev Next > Page: 1 of 1 | 1 Results |

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.

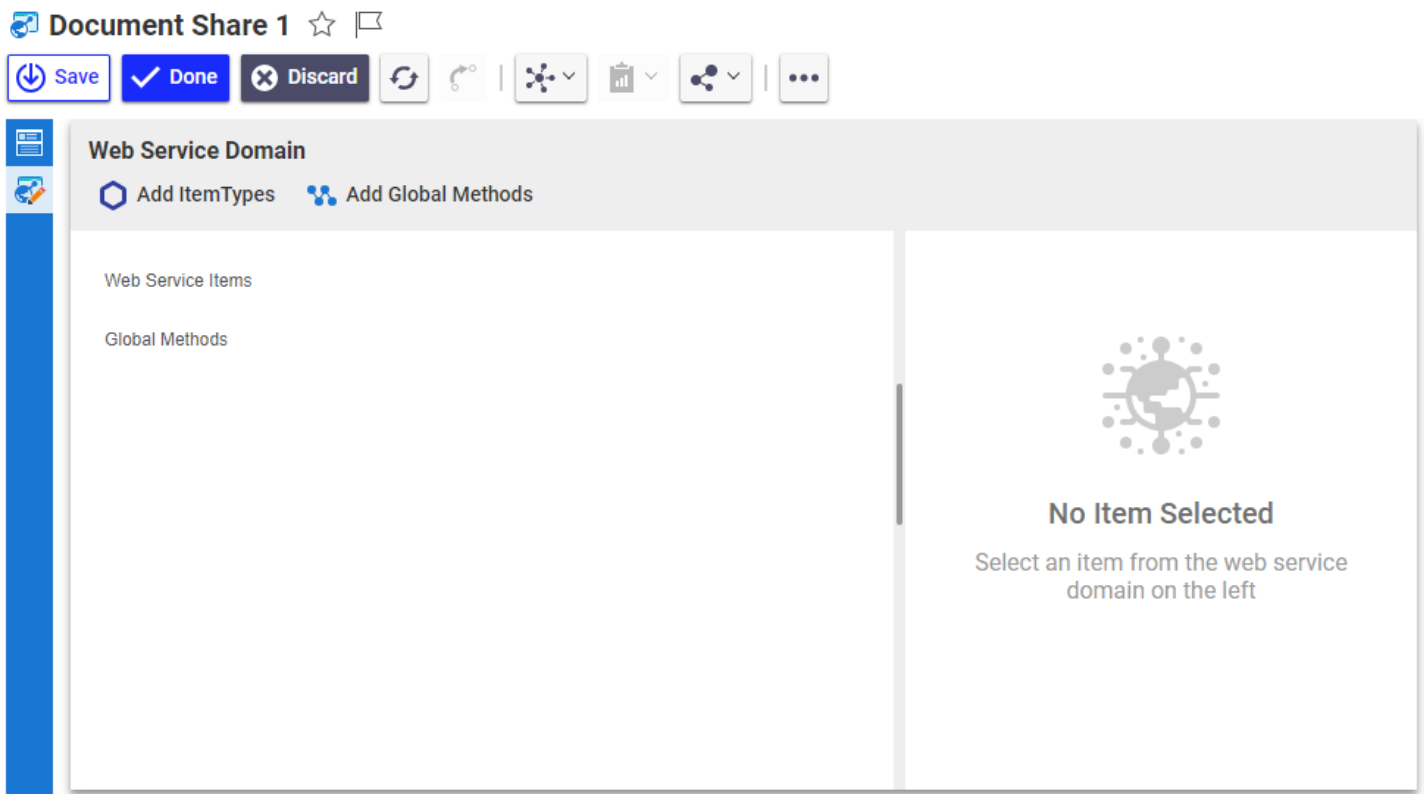
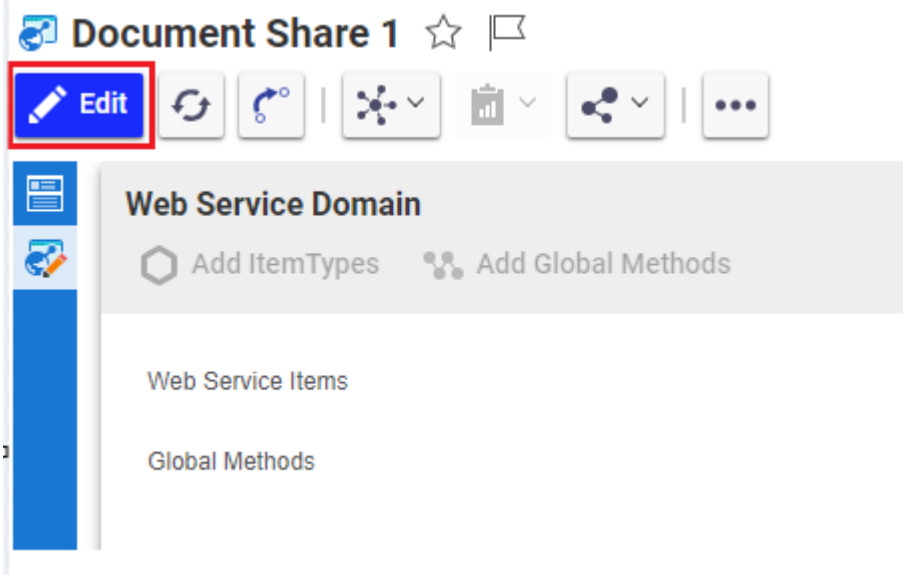




Adding ItemTypes

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.



Select Items



ItemTypes ▾

Simple ▾

	Name ↑	Singular Label	Plural Label	Version...	Depend...	Relatio...	C..	Use Src A...	Descript
▾									



< Prev Next > Page: 1 ...

25

OK

Cancel

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



Select Items

ItemTypes

▼

Q

✕

Simple

▼

	Name ↑	Singular Label	Plural Label	Version...	Depend...	Relatio...	C..	Use Src A...	Descript
▼									
	ac_PolicyAccess...	Attribute Query	Attribute Queries	☐	☐	☒	☐	☒	
	ac_PolicyAccessl...	Derived Attribute ...	Derived Attribute ...	☐	☐	☐	☐	☐	
	Access	Access	Permissions	☐	☐	☒	☒	☒	
	Access_ExplicitD...			☐	☐	☒	☐	☒	

< Prev

Next >

Page: 1

...

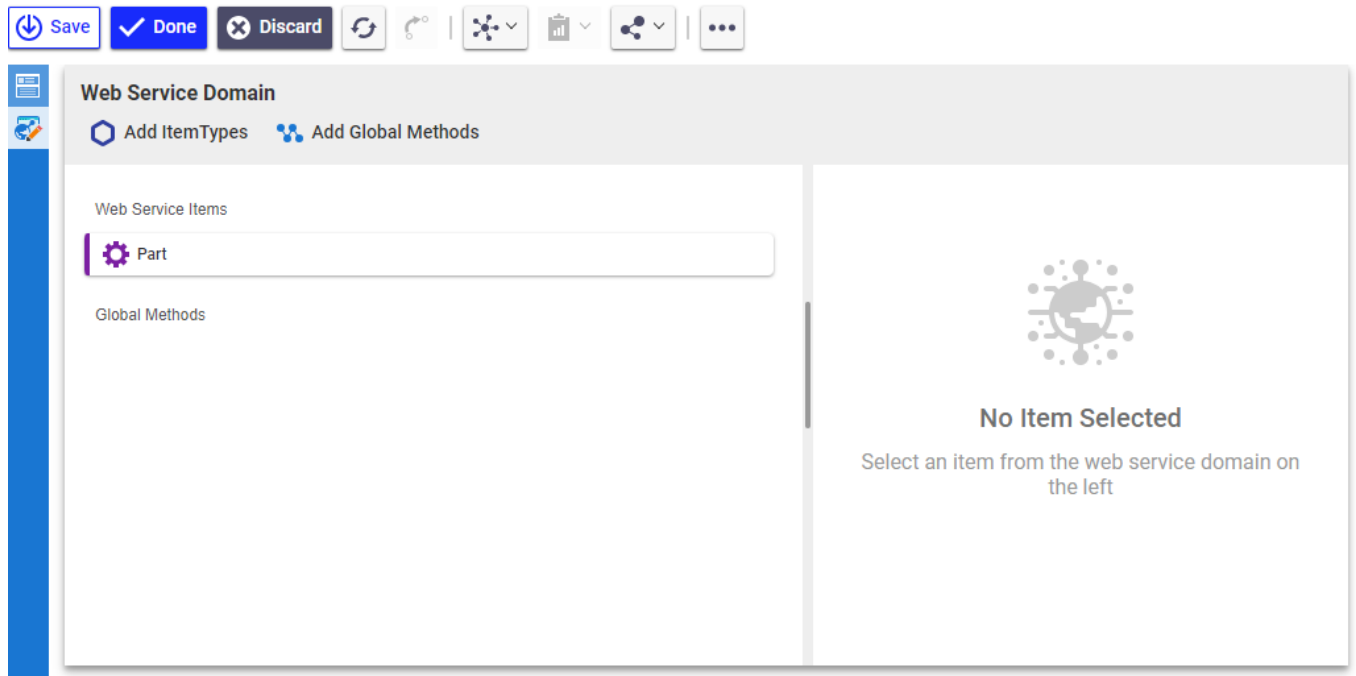
25

Download

OK

Cancel

4. Select the required **ItemType(s)** and click **OK**.
- The selected **ItemType(s)** will be added to the **Web Service Items** list.



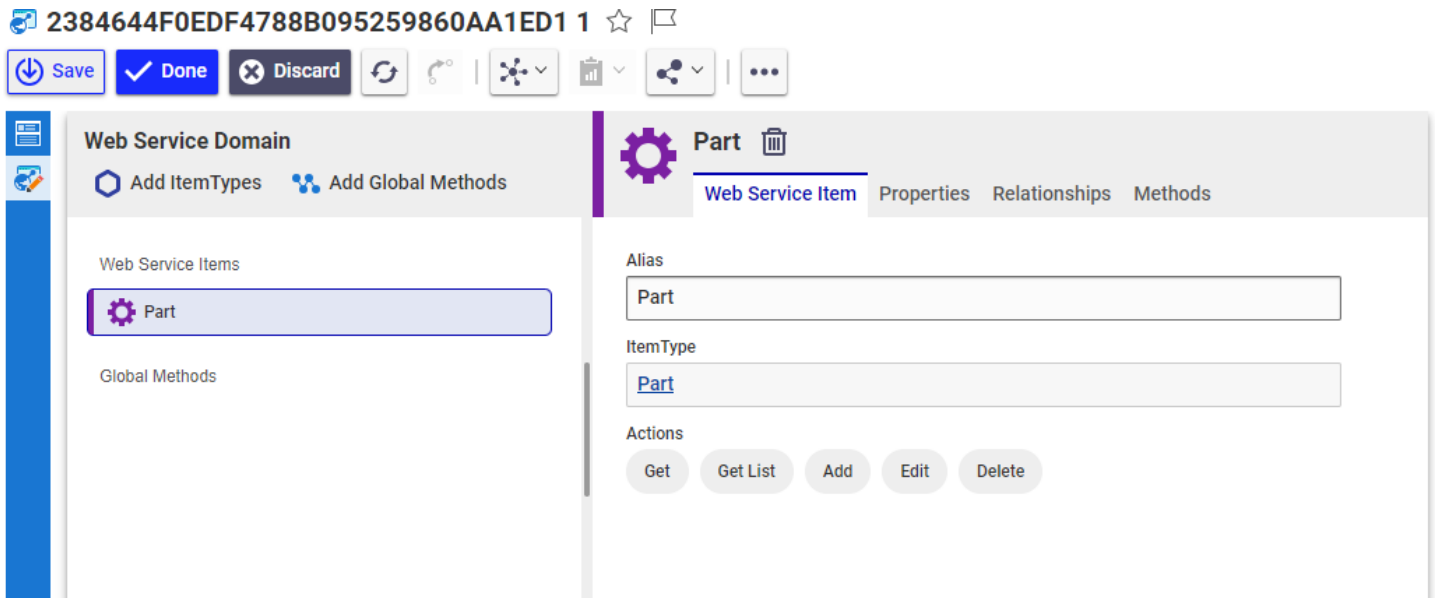
Web Service Items may be removed from the Endpoint by selecting the item in the left pane and clicking the Delete button in the header of the right pane. Removing a Web Service Item from an Endpoint does not delete the ItemType from the database.

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



2384644F0EDF4788B095259860AA1ED1 1 ☆

Save Done Discard

Web Service Domain

Add ItemTypes Add Global Methods

Web Service Items

Part

Global Methods

Part

Web Service Item Properties Relationships Methods

Alias

Part

ItemType

Part

Actions

Get Get List Add Edit Delete

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.

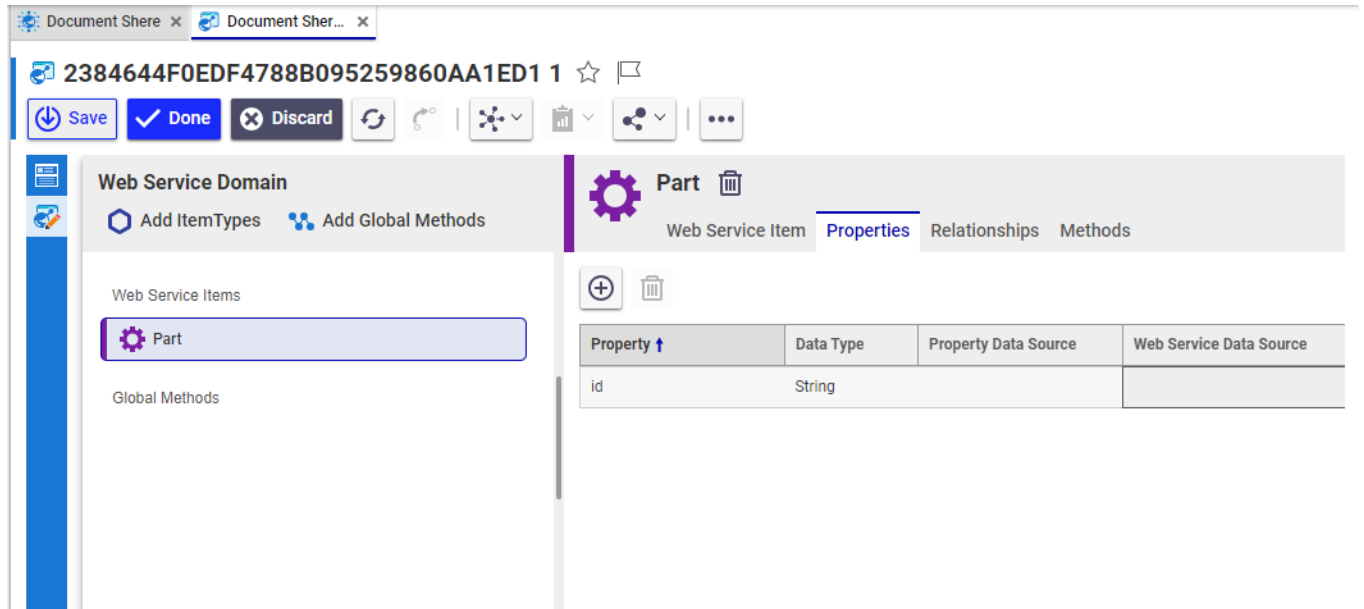
Important

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

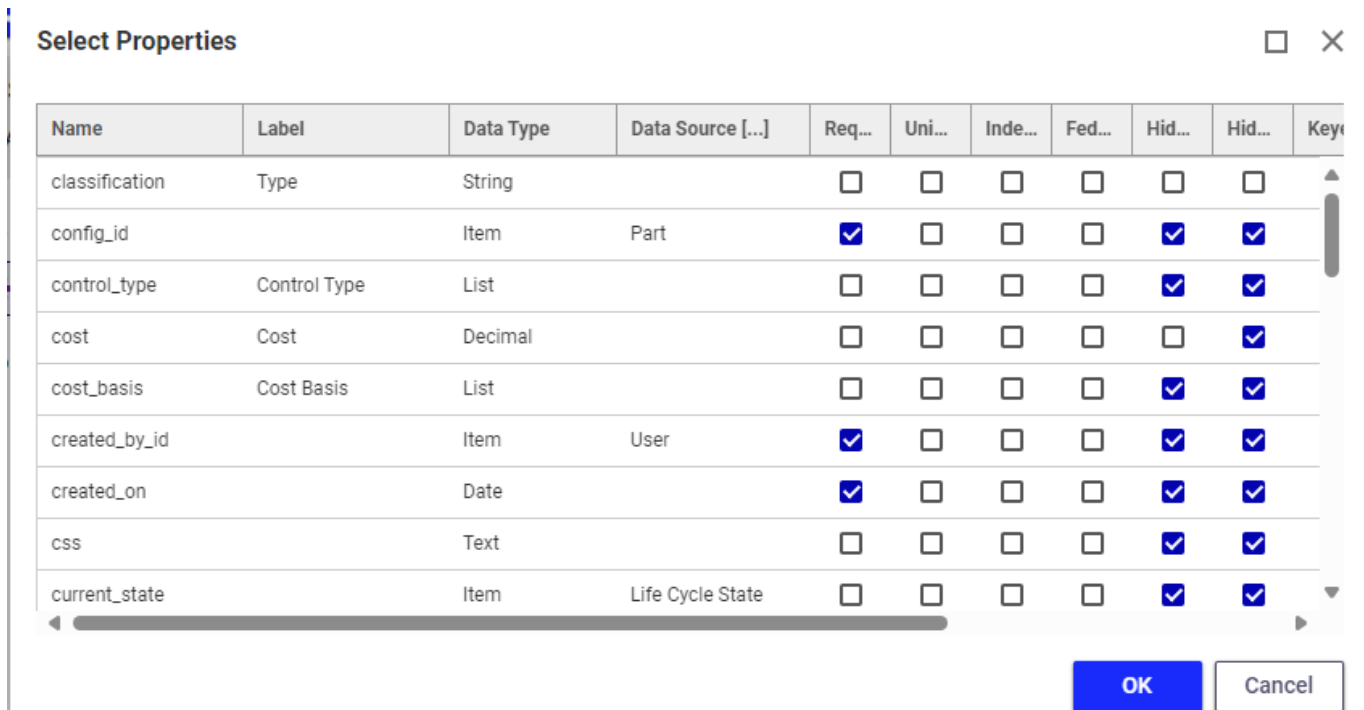
The following steps outline the process to select properties for the Web Service Item:

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.

Select Properties



Name	Label	Data Type	Data Source [...]	Req...	Uni...	Inde...	Fed...	Hid...	Hid...	Key...
classification	Type	String		☐	☐	☐	☐	☐	☐	
config_id		Item	Part	☒	☐	☐	☐	☒	☒	
control_type	Control Type	List		☐	☐	☐	☐	☒	☒	
cost	Cost	Decimal		☐	☐	☐	☐	☐	☒	
cost_basis	Cost Basis	List		☐	☐	☐	☐	☒	☒	
created_by_id		Item	User	☒	☐	☐	☐	☒	☒	
created_on		Date		☒	☐	☐	☐	☒	☒	
css		Text		☐	☐	☐	☐	☒	☒	
current_state		Item	Life Cycle State	☐	☐	☐	☐	☒	☒	

OK

Cancel

5. Click **OK**.

The newly added properties will be displayed as shown below:



**Part**

Web Service Item

Properties

Relationships

Methods



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.



 **Part** 

Web Service Item

Properties

Relationships

Methods





Relationship Alias ↑	Related Alias
----------------------	---------------

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



Add Relationships



Relationship	Related
Part Alternate	Part
Part AML	Manufacturer Part
Part BOM	Part
Part CAD	CAD
Part Changes	
Part Document	Document
Part Goal	
Part MultiLevel BOM	

OK

Cancel

3. Select one or more Relationships.



Add Relationships



Relationship	Related
 Part Alternate	 Part
 Part AML	 Manufacturer Part
 Part BOM	 Part
 Part CAD	 CAD
 Part Changes	
 Part Document	 Document
 Part Goal	
 Part MultiLevel BOM	

4. Click **OK**.

The newly added relationships will be displayed in the table.

 **Part** 

[Web Service Item](#) [Properties](#) [Relationships](#) [Methods](#)

Relationship Alias ↑	Related Alias
Part_AML	Manufacturer_Part
Part_Changes	
Part_Goal	



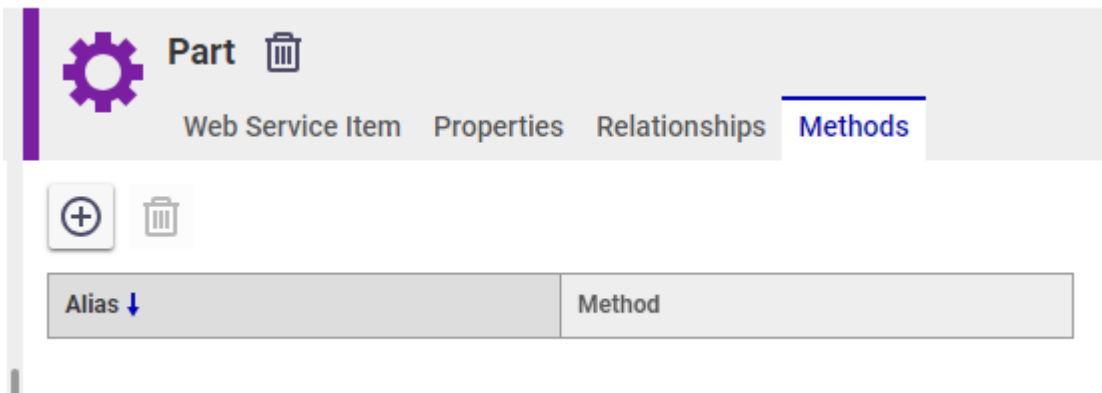
Relationships may be removed from the Web Service Item by selecting one or more rows and clicking the delete button directly above the table. Removing a relationship from a Web Service Item does not affect the ItemType definition.

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**.

Select Items

 **Methods** ▾





Simple ▾

	Name ↑	Method T...	C...	V..	execution_all...	Template [...]	Comments
▾		▾					

A list of items will be displayed.

- 4. Select one or more items.

Select Items

 **Methods** ▾





Simple ▾

	Name ↑	Method T...	C...	V..	execution_all...	Template [...]	Comments
▾	Run	CSharp ▾					
	RunChangeReport	CSharp	☐		Administrators		

- 5. Click **OK**.

The newly added records will be displayed.



Part

Web Service Item
Properties
Relationships
Methods

Alias	Method
RunChangeReport	RunChangeReport
bomrollup	bomrollup

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.

Document Shere 1

Save
 Done
 Discard

Web Service Domain

Add ItemTypes
 Add Global Methods

Web Service Items

Part

Global Methods

Part

Web Service Item
Properties
Relationships
Methods

Alias

Part

ItemType

[Part](#)

Actions

Get
Get List
Add
Edit
Delete



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**.

Select Items

Methods

Simple

	Name	Method T...	C...	V..	execution_all...	Template [...]	Comments

< Prev

Next >

Page: 1

25

OK

Cancel

3. Click **Search**.

A list of all the corresponding Items will be displayed.



Select Items



Methods ▾

 ▾

▼	Name ↑	Method T...	C...	V..	execution_all...	Template [...]	Comments
▼		▼			📄	📄	
	ac_BeforeChangeItemAttri...	Internal	☐	1	Administrators		
	ac_ClearTargetPropOnEdit...	JavaScript	☐	1	Administrators		
	ac_OnBeforeChangeAttrD...	Internal	☐	1	Administrators		
	ac_OnBeforeDeleteAttrDe...	CSharp	☐	1	Administrators		
	ac_PreventChangeApplied...	JavaScript	☐	1	Administrators		

 Page: 1 |

4. Select one or more server Methods.

Select Items



Methods ▾

 ▾

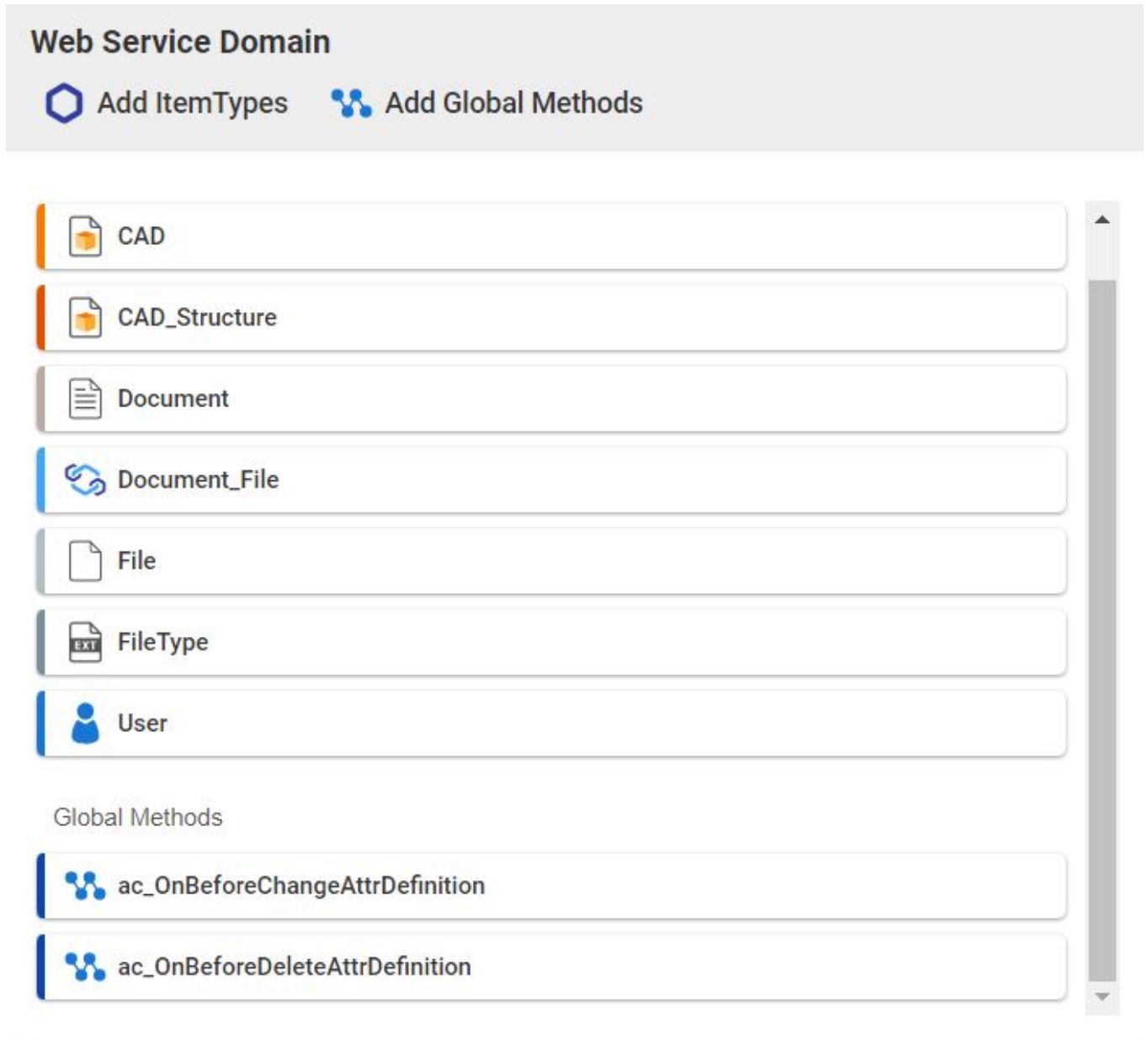
▼	Name ↑	Method T...	C...	V..	execution_all...	Template [...]	Comments
▼		▼			📄	📄	
	ac_BeforeChangeItemAttri...	Internal	☐	1	Administrators		
	ac_ClearTargetPropOnEdit...	JavaScript	☐	1	Administrators		
	ac_OnBeforeChangeAttrD...	Internal	☐	1	Administrators		
	ac_OnBeforeDeleteAttrDe...	CSharp	☐	1	Administrators		
	ac_PreventChangeApplied...	JavaScript	☐	1	Administrators		

 Page: 1 |



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.

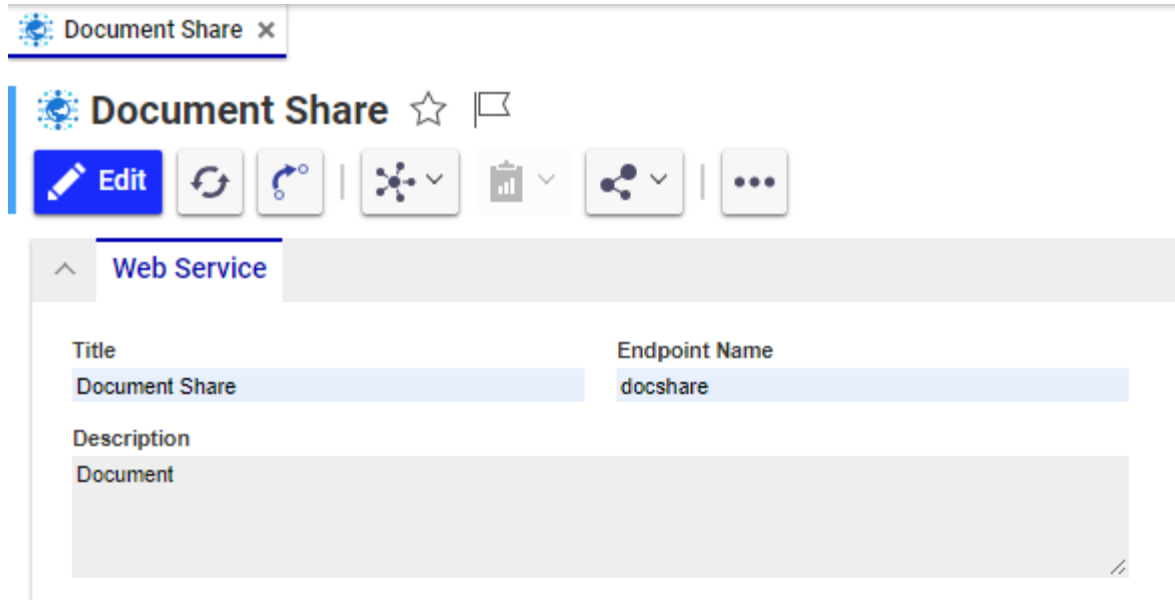
Global Methods may be removed from the Endpoint by selecting the Method in the left pane and clicking the **Delete** button in the header of the right pane. Removing a Global Method from an Endpoint does not delete the Method from the database.

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**.



The screenshot shows the 'Document Share' endpoint configuration interface. At the top, there is a tab labeled 'Document Share' with a close button. Below the tab, the title 'Document Share' is displayed with a star icon and a flag icon. A toolbar contains several icons: a blue 'Edit' button, a refresh icon, a circular arrow icon, a network diagram icon, a bar chart icon, a share icon, and a three-dot menu icon. Below the toolbar, a 'Web Service' section is expanded, showing a table with two columns: 'Title' and 'Endpoint Name'. The table contains one row with 'Document Share' in the 'Title' column and 'docshare' in the 'Endpoint Name' column. Below the table, there is a 'Description' field with the text 'Document'.

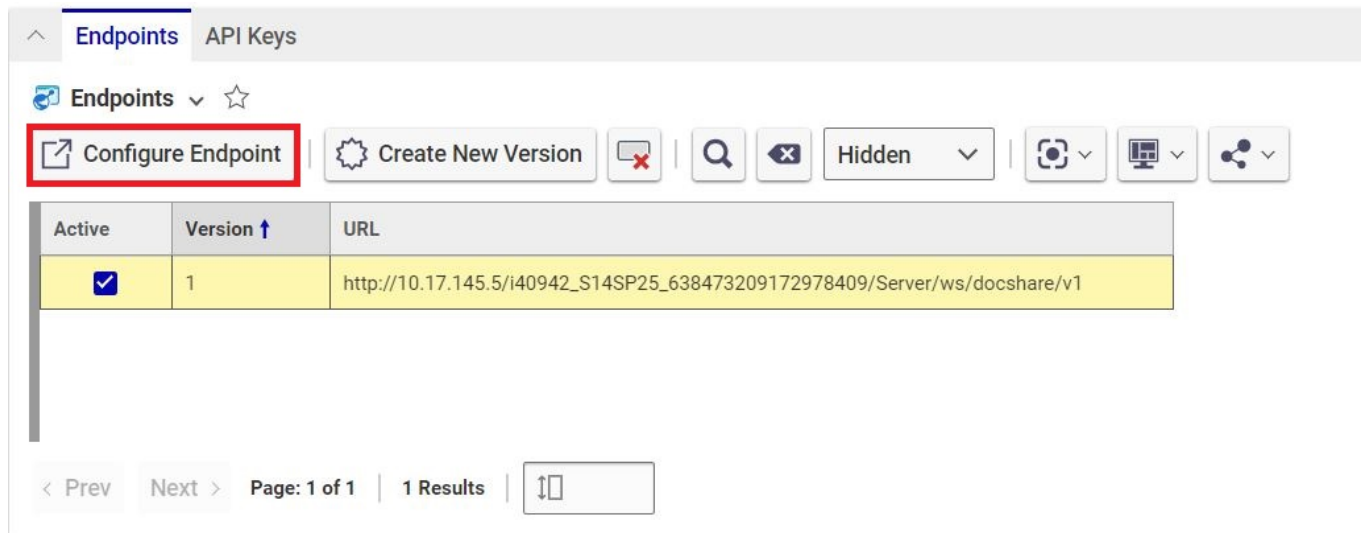
Title	Endpoint Name
Document Share	docshare

Description
Document

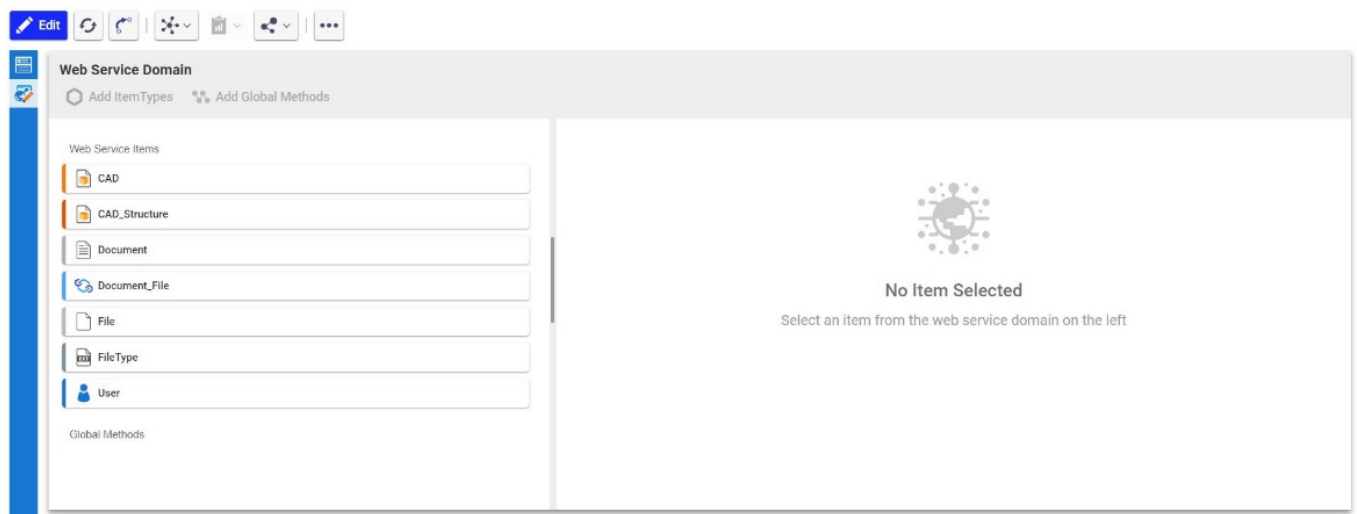
3. Select the Endpoint by clicking on it.



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.

The screenshot displays the 'Web Service' configuration page. At the top, there is a toolbar with buttons for 'Save', 'Done', 'Discard', and several icons for refreshing, undo, redo, and other actions. Below the toolbar, the 'Web Service' section is visible, containing fields for 'Title' (Document Share Service), 'Endpoint Name' (docshare), and 'Description' (This web service shares info about Documents, CAD items, and their files). Below this, the 'Endpoints' tab is selected, showing a table of endpoints. The table has columns for 'Active', 'Version', and 'URL'. A single endpoint is listed with 'Active' checked, 'Version' 1, and a specific URL. The 'Active' checkbox is highlighted with a red box.

Active	Version ↑	URL
☒	1	http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1

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

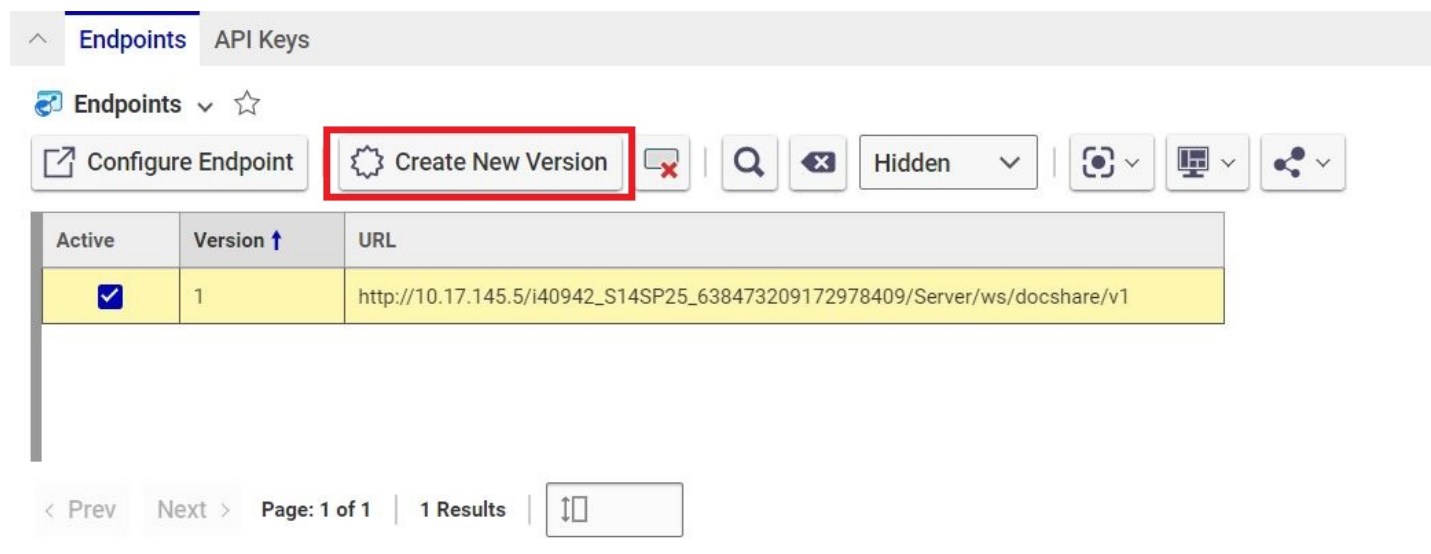
The Active property can also be managed on the Endpoint form.

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.



The screenshot shows the 'Endpoints' tab in a web application. The 'Endpoints' grid is visible, and the 'Create New Version' button in the toolbar is highlighted with a red box. The grid contains one endpoint with the URL 'http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1'.

Active	Version ↑	URL
☒	1	http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1

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.



Endpoints API Keys

Endpoints

Configure Endpoint

Create New Version

Hidden

Active	Version ↑	URL
☒	1	http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1
☒	2	http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v2

< Prev

Next >

Page: 1 of 1

2 Results

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.

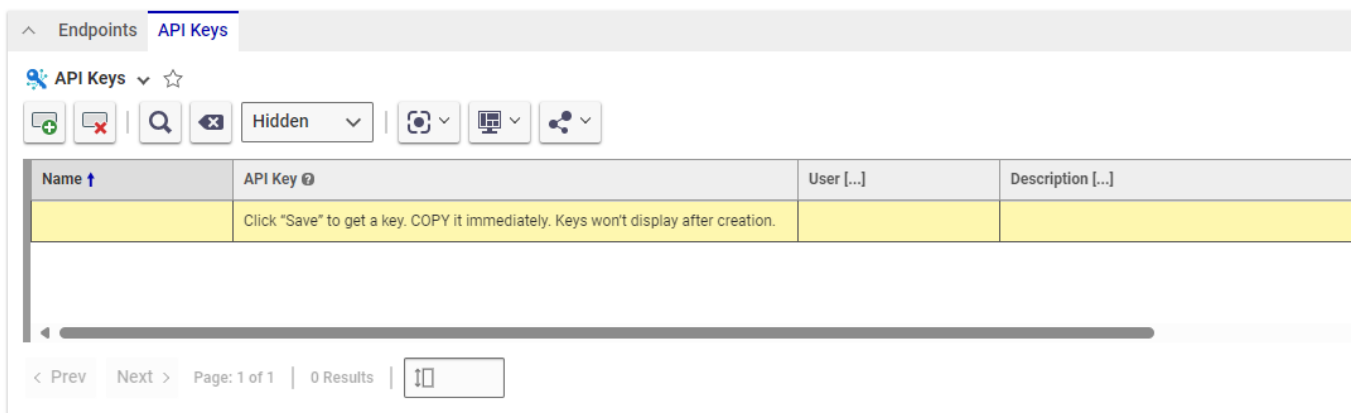
1. Open the Web Service.
2. Click **Edit**.
3. Click the **API Keys** tab.





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.

Name ↑	API Key	User [...]	Description [...]	Created By [...]	Cr
Demo	NDASMDRfU1FMMjAxMI9maXBzLTZyODQ3MDg4Mjc5NDEzMzYxMy43dWJkUj...	Demo Document Share		Super User	4/...

8. Click **Done**.

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

Name ↑	API Key	User [...]	Description [...]	Created By [...]	Cr
Demo	NDASMDRfU1FMMjAxMI9maXBzLTZyODQ3MDg4Mjc5NDEzMzYxMy43dWJkUj...	Client Admin		Super User	4/...

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.

EndpointsAPI Keys

Endpoints

Configure Endpoint

Create New Version

Hidden

Active	Version	URL ↑
☒	1	http://10.17.144.173/i40904_S14SP26_638470880416940874/Server/ws/docshare/v1

< Prev

Next >

Page: 1 of 1

1 Results

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.

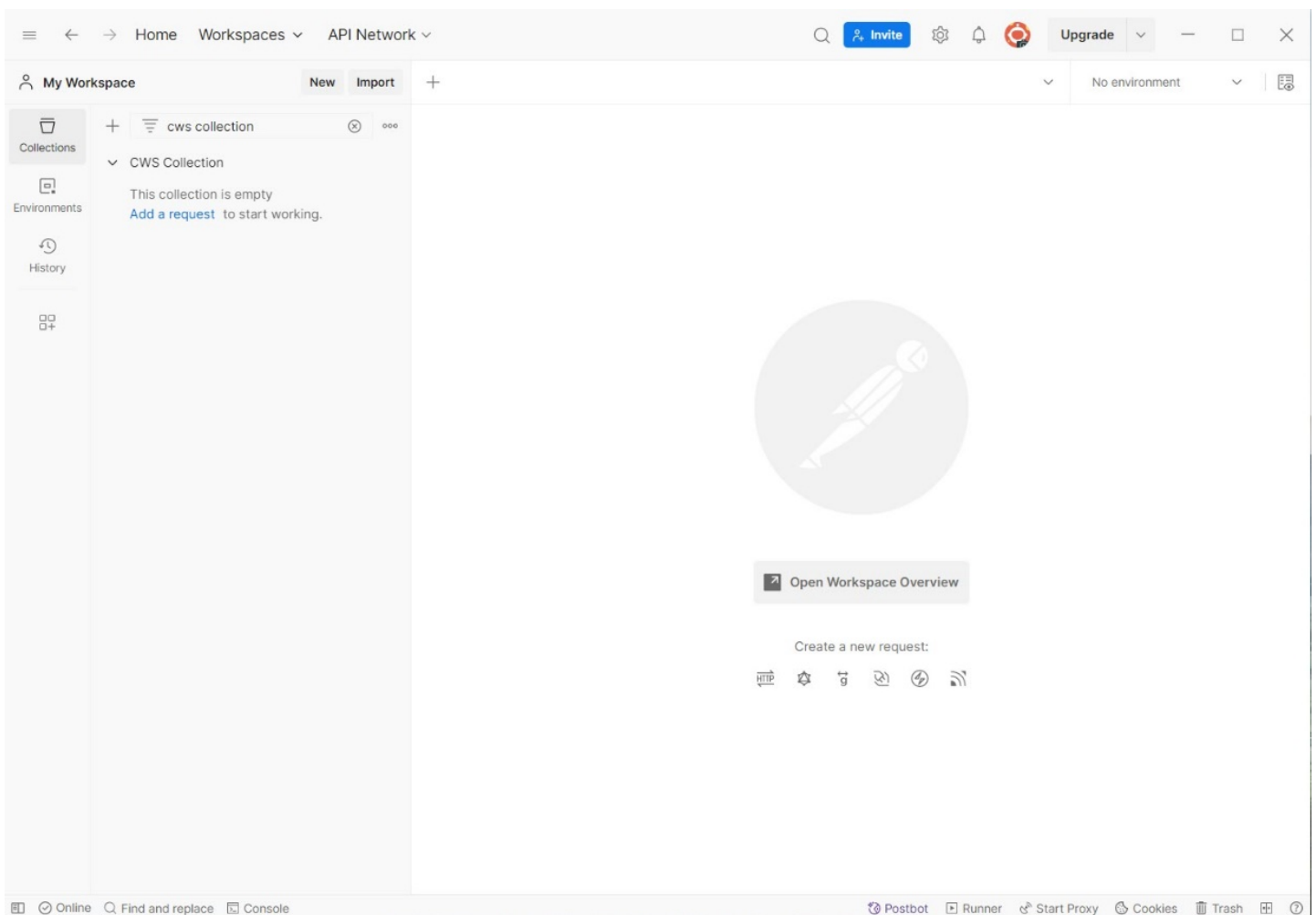
API Key

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

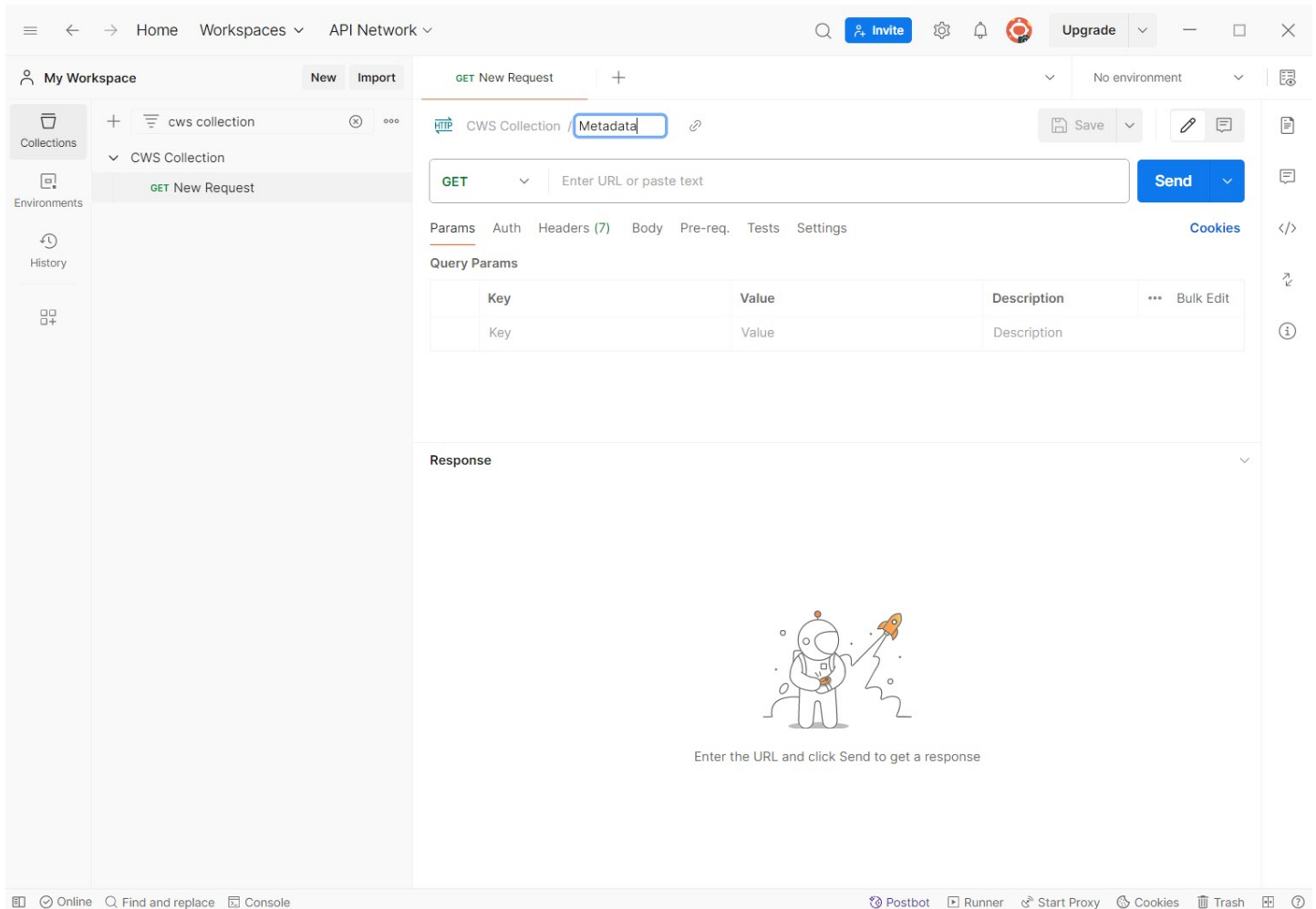
Important

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**.



Home Workspaces API Network

My Workspace New Import

Collections + cws collection

Environments

History

GET Metadata

GET Metadata

CWS Collection / Metadata

GET http://10.17.145.5/40942_S14SP25_638473209172978409/Server/ws/docshare/v1

Params Auth Headers (8) Body Pre-req. Tests Settings Cookies

Query Params

Key	Value	Description	Bulk Edit
Key	Value	Description	

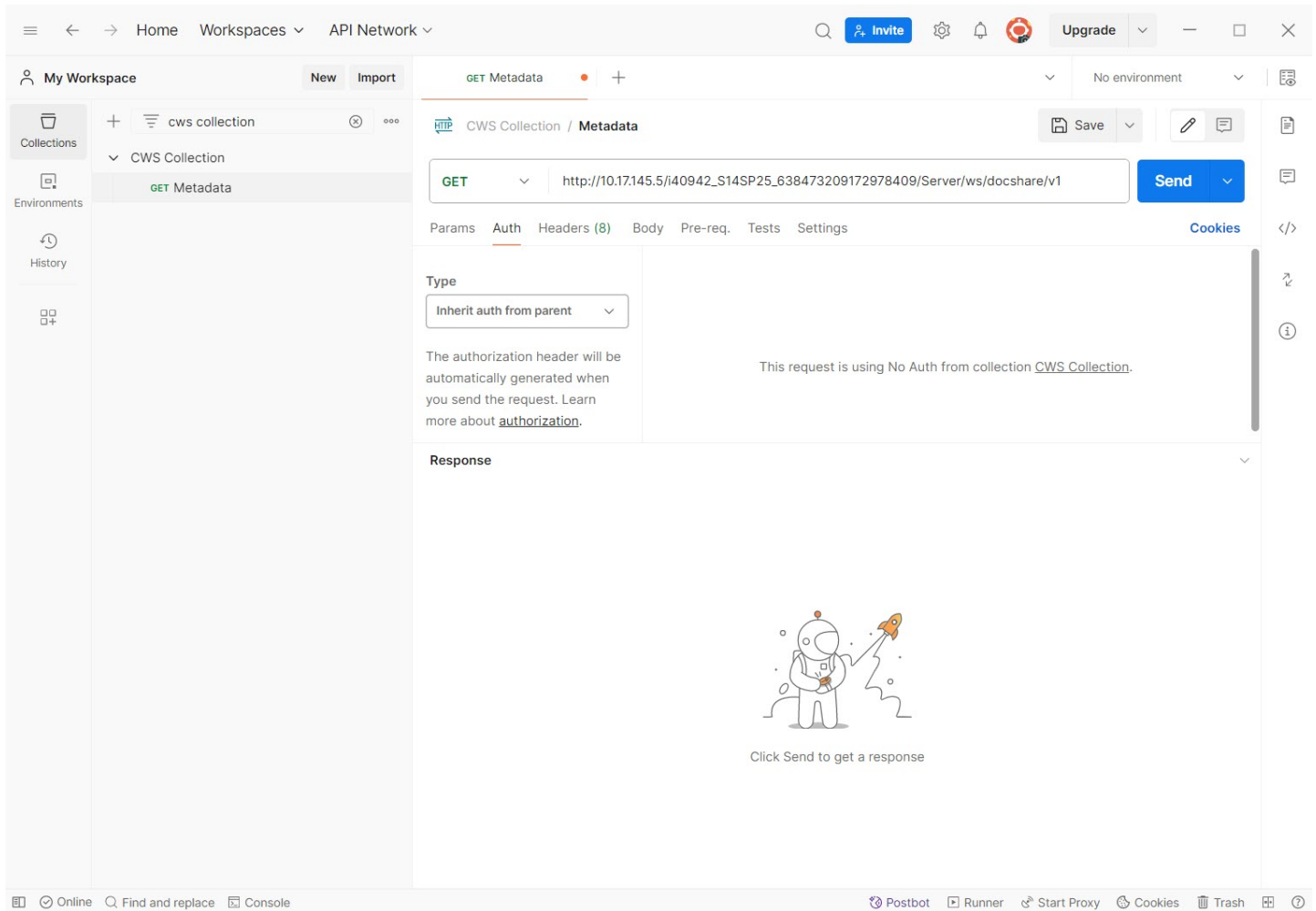
Response

Click Send to get a response

Online Find and replace Console Postbot Runner Start Proxy Cookies Trash

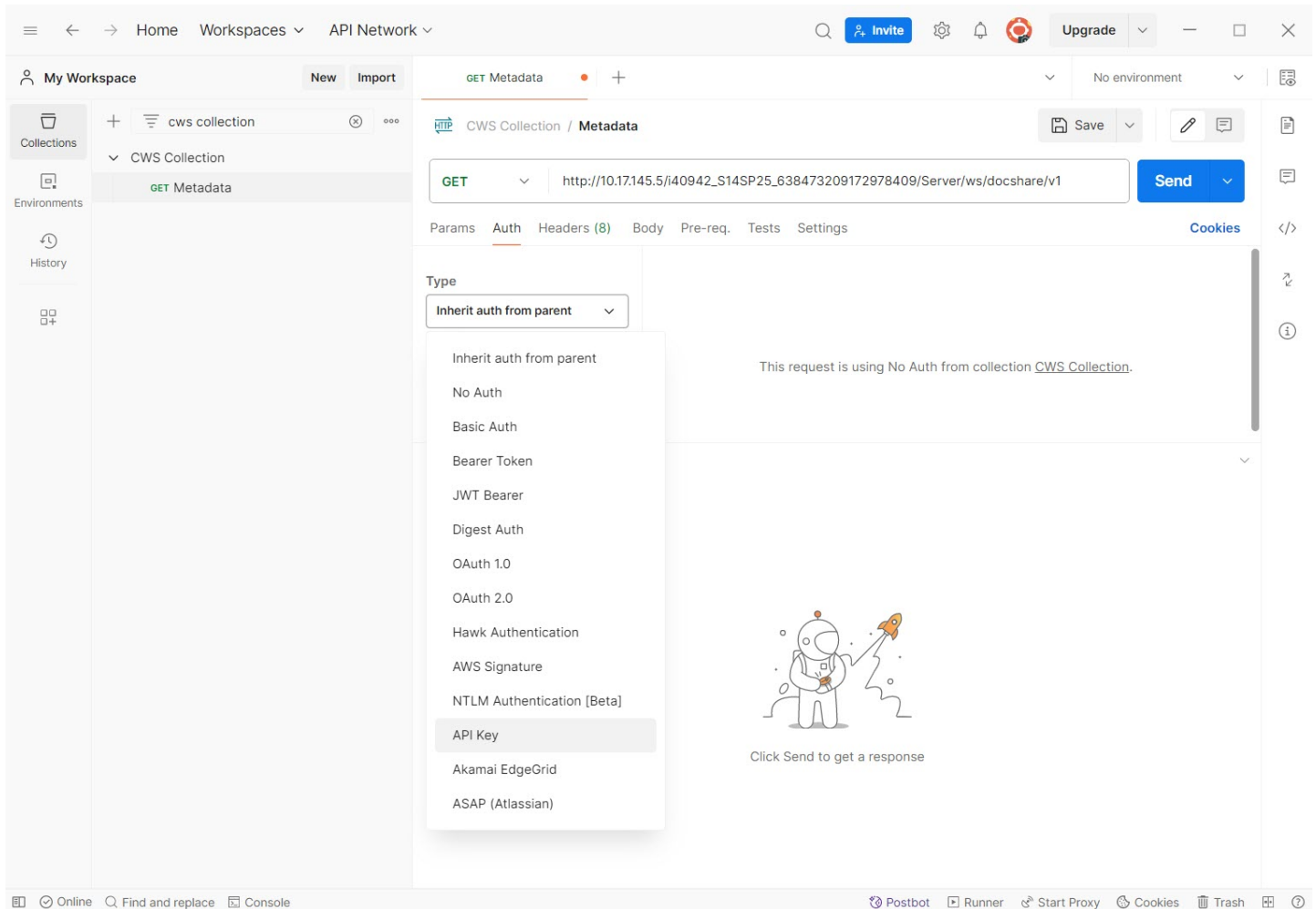
7. Click the **Auth** tab.





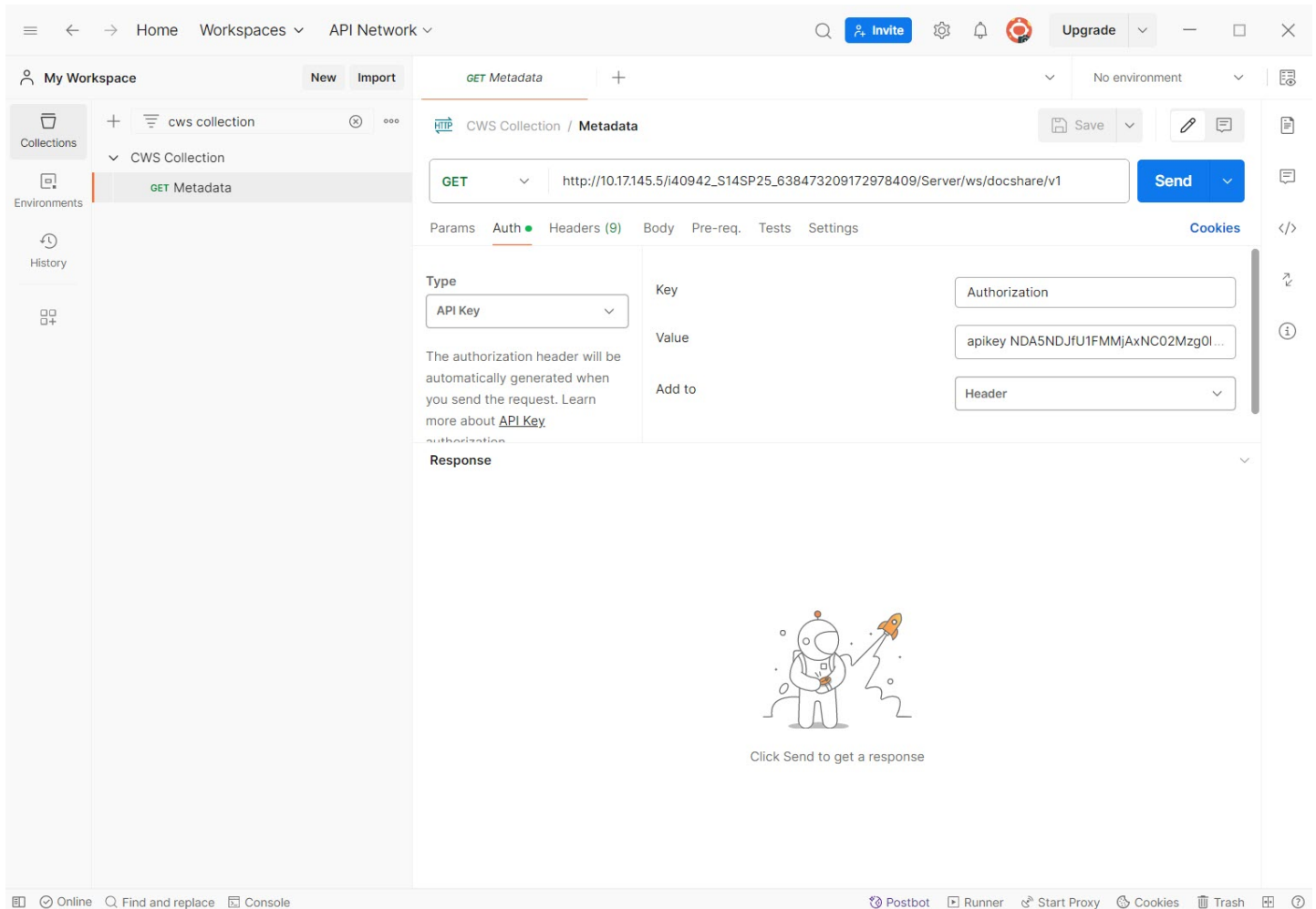
8. Select **API Key** from the **Type** dropdown.





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.



The screenshot shows the API Network interface with a GET request to the endpoint `http://10.17.145.5/40942_S14SP25_638473209172978409/Server/ws/docshare/v1`. The request is configured with an API Key in the Authorization header. The response is displayed in JSON format, showing an Odata-compliant schema with entities like Document, CAD, and File.

```

1 {
2   "@odata.context": "http://10.17.145.5/40942_S14SP25_638473209172978409/Server/ws/docshare/
3     $metadata",
4   "value": [
5     {
6       "name": "Document",
7       "kind": "EntitySet",
8       "url": "Document"
9     },
10    {
11      "name": "CAD",
12      "kind": "EntitySet",
13      "url": "CAD"
14    },
15    {
16      "name": "File",
17      "kind": "EntitySet",
18      "url": "File"
19    }
20  ]
21 }

```

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.



The screenshot shows the Postman API client interface. At the top, the breadcrumb navigation indicates 'Home' > 'Workspaces' > 'API Network'. The main header shows the request method 'GET' and the endpoint 'Metadata' under the 'CWS Collection'. The URL bar contains the full endpoint: `http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1/#metadata`. The 'Authorization' tab is selected, showing an 'API Key' type with the key 'Authorization' and the value 'apikey NDA5NDJfU1FMMjAxNC02Mzg0NzI...'. The 'Body' tab is also visible, showing a JSON response in 'Pretty' format. The response status is '200 OK' with a time of '200 ms' and a size of '1.05 KB'. The JSON response is as follows:

```

1 {
2   "@odata.context": "http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1/$metadata",
3   "value": [
4     {
5       "name": "Document",
6       "kind": "EntitySet",
7       "url": "Document"
8     },
9     {
10      "name": "CAD",
11      "kind": "EntitySet",
12      "url": "CAD"
13    },
14    {
15      "name": "File",
16      "kind": "EntitySet",
17      "url": "File"
18    },
19    {
20      "name": "Image",
21      "kind": "EntitySet",
22      "url": "Image"
23    }
24  ]
25 }

```

At the bottom of the interface, there are utility buttons for 'Online', 'Find and replace', 'Console', 'Postbot', 'Runner', 'Start Proxy', 'Cookies', 'Trash', and a help icon.

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.



The screenshot shows the Aras API Network interface. The top bar includes navigation links (Home, Workspaces, API Network) and a search bar. The main area displays a GET request to the endpoint `http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1/Document`. The response status is 200 OK, with a time of 104 ms and a size of 1.92 KB. The response body is shown in JSON format, containing an array of two document objects.

```

1 {
2   "@odata.context": "http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1/$metadata#Document",
3   "value": [
4     {
5       "id": "10D94C0CA6404F799F3E3F88A5801256",
6       "classification": "Product",
7       "state": "Preliminary",
8       "major_rev": "A",
9       "name": "Configurable User Interface Administrator Guide",
10      "has_files": "1",
11      "item_number": "D-000200",
12      "modified_on": "2024-04-05T21:00:52",
13      "keyed_name": "D-000200",
14      "generation": 1,
15      "created_on": "2024-04-05T20:59:24"
16    },
17    {
18      "id": "6D9D7A13C35A4801A27EF70E98386880",
19      "classification": null,
20      "state": "Preliminary",
21      "major_rev": "A",
22      "name": "Configurator Services Programmers Guide",
23      "has_files": "0"
24    }
25  ]
26 }

```

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.



The screenshot shows the Postman interface with a GET request to the endpoint `http://10.17.145.5/i40942_S14SP25_638473209172978409/Server/ws/docshare/v1/Part`. The request is saved and the response is displayed in the Body tab as a JSON error message:

```

1 {
2   "error": {
3     "code": 400,
4     "message": "Resource not found for the segment 'Part'."
5   }
6 }

```

The status bar indicates a 400 Bad Request with a response time of 197 ms and a size of 675 B. The bottom of the interface shows various toolbars including Online, Find and replace, Console, Postbot, Runner, Start Proxy, Cookies, Trash, and a help icon.

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).



POST ⌵ {{ServiceRoot}}/\$batch Send ⌵

Params Authorization Headers (14) Body Pre-request Script Tests Settings Cookies

Headers ⌵ 10 hidden

Key	Value	Description	...	Bulk Edit	Presets
☒ DATABASE	{{DATABASE}}				
☒ AUTHUSER	{{AUTHUSER}}				
☒ AUTHPASSWORD	{{AUTHPASSWORD}}				
☒ Content-Type	multipart/mixed; boundary=batch_abcdeff8-fc75-4b56-8c71-56071383e77b				
Key	Value	Description			

- **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.

The screenshot shows a REST client interface with a POST request to `http://{{HOST}}/{{alias}}/Server/ws/CWS_OdataSet/v1/$batch`. The 'Body' tab is selected, and the format is set to 'raw'. The request body is a batch of three HTTP requests, each separated by an empty line. Red arrows point to specific parts of the batch with labels:

- Line 1: `--batch_abcdeff8-fc75-4b56-8c71-56071383e77b` (first boundary)
- Line 4: `GET http://{{HOST}}/{{alias}}/Server/ws/EndpointName/v1/Part HTTP/1.1` (absolute URI with full path)
- Line 10: `POST {{alias}}/Server/ws/EndpointName/v1/Part HTTP/1.1` (absolute resource path and separate Host header)
- Line 11: `Host: {{HOST}}`
- Line 23: `DELETE Part('01AA112937924D5383FB4A101B2AFF3E') HTTP/1.1` (resource path relative to the batch request URI)
- Line 26: `--batch_abcdeff8-fc75-4b56-8c71-56071383e77b--` (last boundary)

The batch body includes a JSON payload for the POST request:

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

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).

```

1  --batch_abcodeff8-fc75-4b56-8c71-56071383e77b
2  Content-Type: application/http
3
4  GET Part HTTP/1.1
5
6
7  --batch_abcodeff8-fc75-4b56-8c71-56071383e77b
8  Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
9
10 --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
11 Content-Type: application/http
12 Content-Transfer-Encoding: binary
13 Content-ID: 1
14
15 POST Part HTTP/1.1
16 Content-Type: application/json
17
18 {
19   "id": "01AA112937924D5383FB4A101B2AFF3E",
20   "item_number": "Part-01",
21   "name": "Part-01"
22 }
23
24 --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
25 Content-Type: application/http
26 Content-ID: 2
27
28 POST Part HTTP/1.1
29 Content-Type: application/json
30
31 {
32   "id": "00BB112937924D5383FB4A101B2AFF56",
33   "item_number": "Part-00",
34   "name": "Part-00",
35   "Part-BCM": [
36     {
37       "id": "00E01E12F3004B3283F3C88B9349E4A0",
38       "related_id@odata.bind": "$1"
39     }
40   ]
41 }
42
43 --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
44 Content-Type: application/http
45 Content-ID: 3
46
47 DELETE $2 HTTP/1.1
48
49
50 --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
51
52 --batch_abcodeff8-fc75-4b56-8c71-56071383e77b
53 Content-Type: application/http
54
55 DELETE Part('01AA112937924D5383FB4A101B2AFF3E') HTTP/1.1
56
57
58 --batch_abcodeff8-fc75-4b56-8c71-56071383e77b--

```

change set boundary

first change set request

second change set request

third change set request

last boundary (end of change set)

- 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.

```

1  --batch_abcddef8-fc75-4b56-8c71-56071383e77b
2  Content-Type: multipart/mixed; boundary=changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
3
4  --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
5  Content-Type: application/http
6  Content-ID: :1
7
8  POST Part: HTTP/1.1
9  Content-Type: application/json
10
11  {
12    → "id": "01AA112937924D5383FB4A101B2AFF3E",
13    → "item_number": "Part-01",
14    → "name": "Part-01"
15  }
16
17  --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
18  Content-Type: application/http
19  Content-ID: :1
20
21  PUT Part('01AA112937924D5383FB4A101B2AFF3E')/name: HTTP/1.1
22  Content-Type: application/json
23
24  {
25    → "value": "New-Part-1"
26  }
27
28  --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
29  Content-Type: application/http
30  Content-ID: :2
31
32  GET Part:$1: HTTP/1.1
33
34  --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
35  Content-Type: application/http
36
37  DELETE $1: Part: HTTP/1.1
38
39
40
41  --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd
42  Content-Type: multipart/mixed; boundary=changeset_98357add-c8fa-41ac-a9f8-9357efabc
43
44  --changeset_98357add-c8fa-41ac-a9f8-9357efabc
45  Content-Type: application/http
46  Content-ID: :4
47
48  POST Part: HTTP/1.1
49  Content-Type: application/json
50
51  {
52    → "id": "51BC113537924D5383FB4A101B2AAA3B",
53    → "item_number": "Part-02",
54    → "name": "Part-02"
55  }
56
57  --changeset_98357add-c8fa-41ac-a9f8-9357efabc--
58  --changeset_77162fcd-b8da-41ac-a9f8-9357efbbd--
59
60
61  --batch_abcddef8-fc75-4b56-8c71-56071383e77b--

```

correct Change Set request

Content-ID is not unique

wrong Change Set request

GET HTTP request is not allowed

wrong Change Set request

There is no specified Content-ID

wrong Change Set request

nested Change Set request

wrong Change Set request

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

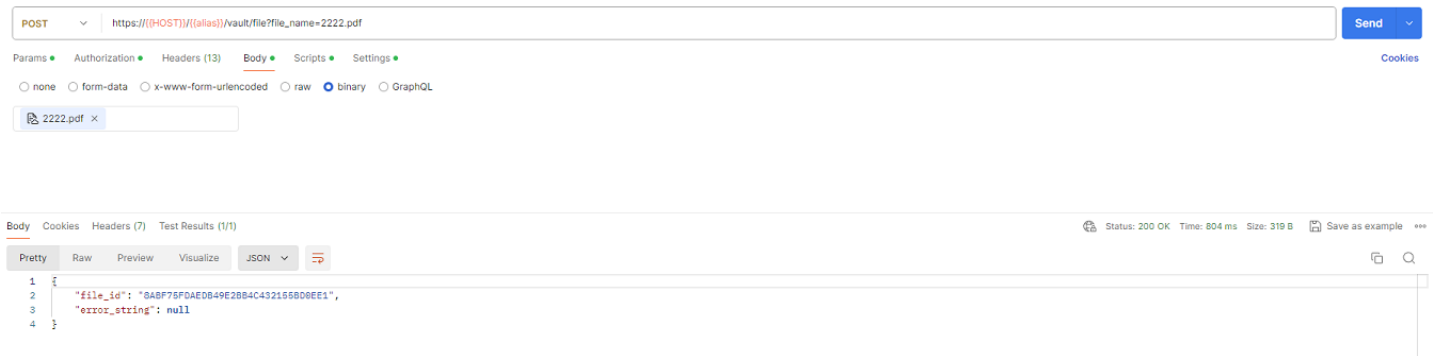
The following steps outline the process of uploading a file to the vault's upload endpoint.

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.

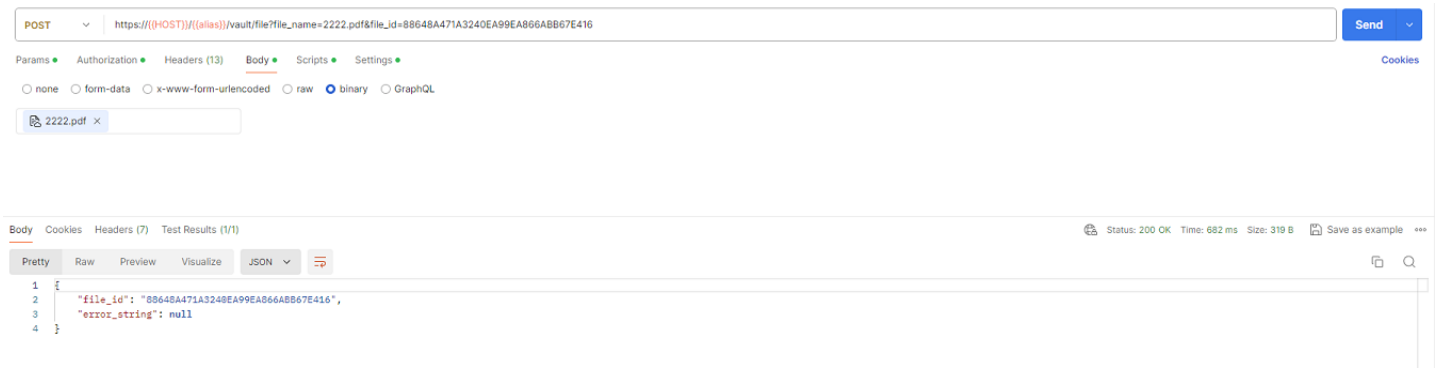


Important

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')/FileNames
```

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}&fileId=22602CF88CC94932A2523585EB5FDB21&fileName=MP0101"
}

```

