**Metadata Editor API – UI Screen to API Mapping Document**

All Pages - Header



The header section displays the user information, which can be retrieved by calling GET /me endpoint.

* 1. These are the initials of the name (See #2).
* 2. This is the ‘name’ property in the response.
* 3. This is the ‘role’ property in the response.

All Pages – Quick Search



When performing the search, the endpoint to use will depend on the type of search being performed:

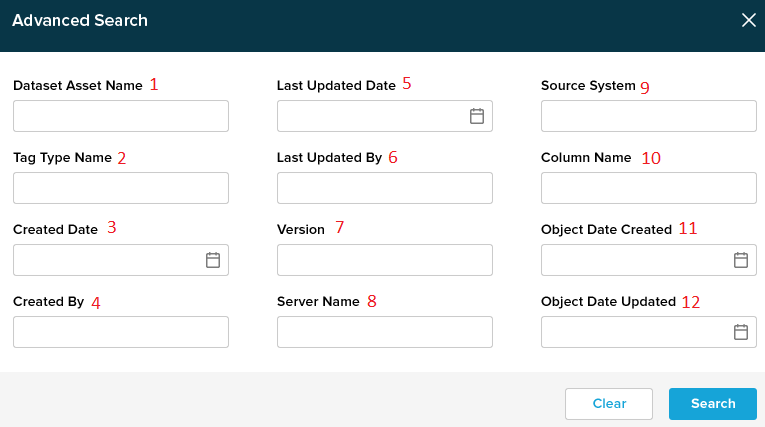
* For Datasets, the POST /datasets/search endpoint is called. The frontend will send the search string as the ‘datasetAssetName’.
* For Tags, the POST /tags/search endpoint is called. The frontend will send the search string as ‘tagTypeNameFilterValue’. The ‘tagTypeNameFilterType’ should be ‘Contains’.
* For All the POST /quicksearch endpoint is called. The frontend will send the search string as the ‘searchString’.



The search results depend on the endpoint that was called:

* The ‘name’ property in the results body is used to display the name of each result.
* The ‘id’ property in the results body is used to generate a link to the respective entity.
* The ‘type’ will depend on the type of search being conducted. For ‘Dataset’ and ‘Tag’ searches, the type is the respective search. For ‘All’ searches, use the ‘type’ property in the results body.

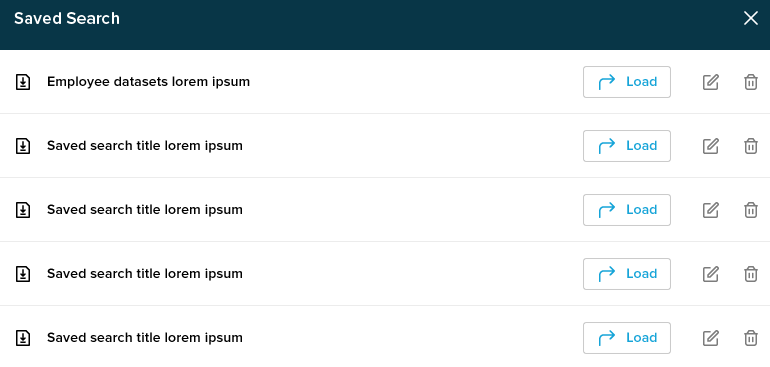
All Pages – Advanced Search



For advanced search, two POST calls are made – one to /datasets/search and another to /tags/search. The search parameters are provided in the request body as follows:

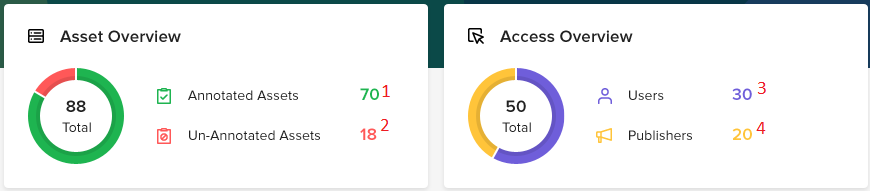
* 1. Corresponds to ‘datasetAssetName’. If this property is provided, then no need to provide a POST call to /tags/search.
* 2. Corresponds to ‘tagTypeNameFilterValue’. The ‘tagTypeNameFilterType’ is set to ‘Contains’.
* 3. Corresponds to ‘createdDateMin’ and ‘createdDateMax’.
* 4. Corresponds to ‘createdBy’.
* 5. Corresponds to ‘lastUpdatedDateMin’ and ‘lastUpdatedDateMax’.
* 6. Corresponds to ‘lastUpdatedBy’.
* 7. Corresponds to ‘version’.
* 8. Corresponds to ‘serverName’.
* 9. Corresponds to ‘sourceSystem’.
* 10. For now, the property is unsupported. (Can be omitted/ignored)
* 11. For now, the property is unsupported. (Can be omitted/ignored)
* 12. For now, the property is unsupported. (Can be omitted/ignored)
* 13. The results are displayed in the Search Results section.

All Pages – Advanced Search



The frontend is responsible for saving and storing the saved search parameters in local storage.

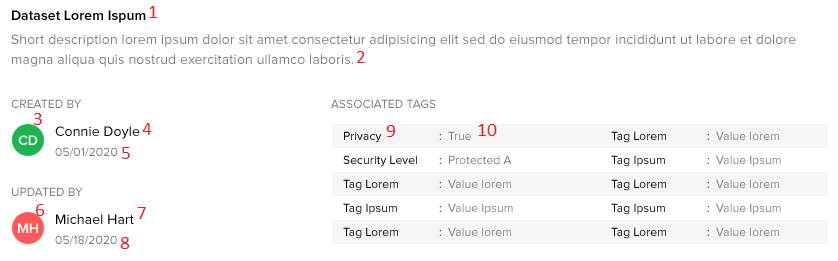
Dashboard – Stats



The dashboard page displays some statistics about the overall status of the application, which can be retrieved by calling GET /dashboardStats. The ring graphs can be created from the data provided in the endpoints as well.

* 1. This corresponds to ‘annotatedAssets’ property.
* 2. This corresponds to ‘unannotatedAssets’ property.
* 3. This corresponds to ‘users’ property.
* 4. This corresponds to ‘publishers’ property.

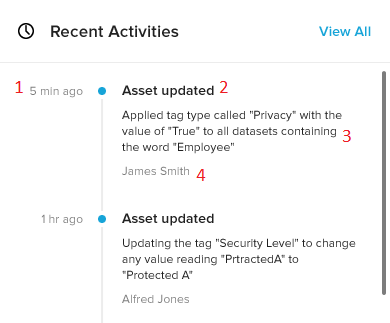
Dashboard – Pinned Assets



The frontend is responsible for storing the list of Pinned Assets. The actual asset information can be retrieved by calling POST /datasets/search with an array of ‘ids’ containing the pinned assets.

* 1. This corresponds to ‘name’ property.
* 2. This corresponds to ‘description’ property.
* 3. This corresponds to initials of ‘createdBy’ property.
* 4. This corresponds to ‘createdBy’ property.
* 5. This corresponds to ‘createdOn’ property.
* 6. This corresponds to initials of ‘updatedBy’ property.
* 7. This corresponds to ‘updatedBy’ property.
* 8. This corresponds to ‘updatedOn’ property.
* 9. Each entry corresponds to ‘tags[n].tagName’ property.
* 10. Each entry corresponds to ‘tags[n].tagValue’ property.

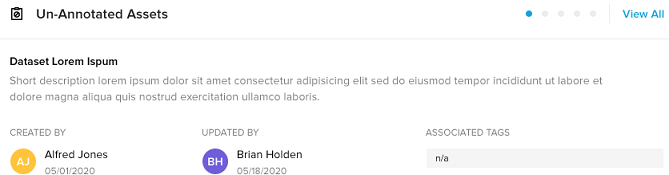
Dashboard – Recent Activity



The recent activities can be retrieved by calling GET /activities. The ‘perPage’ parameter can be used to indicate the size of activities to retrieve. ‘sortBy’ should be by date and ‘sortOrder’ should be in ‘DESC’ order.

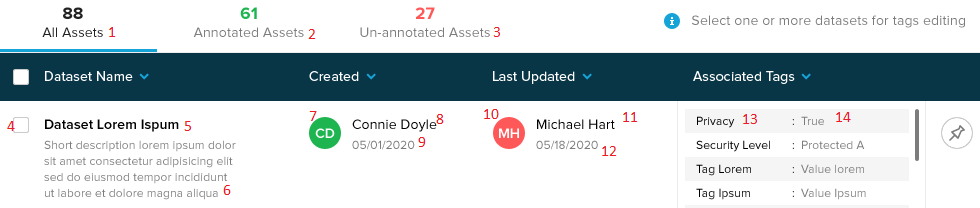
* 1. This corresponds to ‘date’ property.
* 2. This corresponds to ‘type’ property (it might be necessary to map the type to a more readable string for the frontend).
* 3. This corresponds to a string that is generated based on ‘type’, ‘propertyName’, ‘tagName’, ‘oldValue’ and ‘newValue’. See section on Activity for more details.
* 4. This corresponds to ‘actor’ property.

Dashboard – Unannotated Assets



The unannotated Assets can be retrieved by calling POST /datasets/search with ‘isAnnotated’ as ‘false’. The mapping is like the ‘Dashboard – Pinned Assets’ section (with the exception that the Associated Tags no longer need to be displayed).

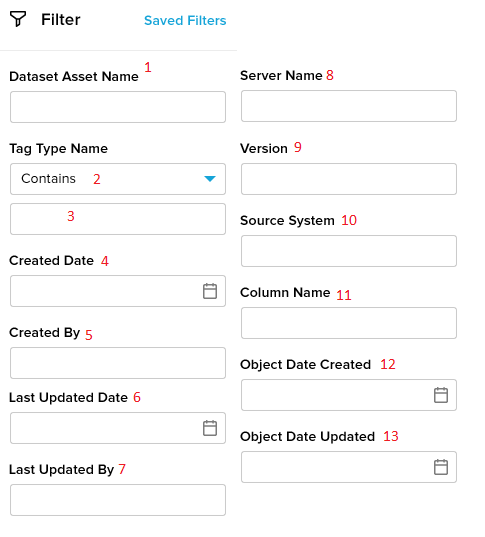
Asset/Dataset Page – Asset List



The initial list of Datasets can be retrieved by providing a blank filter to the POST /datasets/search. Additional filtering is possible (see next section). The API response is used to display the list:

* 1. This corresponds to ‘totalAssets’ property.
* 2. This corresponds to ‘annotatedAssets’ property.
* 3. This corresponds to ‘totalAssets’ – ‘annotatedAssets’.
* 4. The id is used to identify the data entry for other operations.
* 5. This corresponds to ‘name’ property.
* 6. This corresponds to ‘description’ property.
* 7. This corresponds to initials of ‘createdBy’ property.
* 8. This corresponds to ‘createdBy’ property.
* 9. This corresponds to ‘createdOn’ property.
* 10. This corresponds to initials of ‘updatedBy’ property.
* 11. This corresponds to ‘updatedBy’ property.
* 12. This corresponds to ‘updatedOn’ property.
* 13. This corresponds to ‘tags[n].tagName’ property.
* 14. This corresponds to ‘tags[n].tagValue’ property.
* Page size and Page Number can also be controlled via ‘pageSize’ and ‘pageNumber’ parameters.

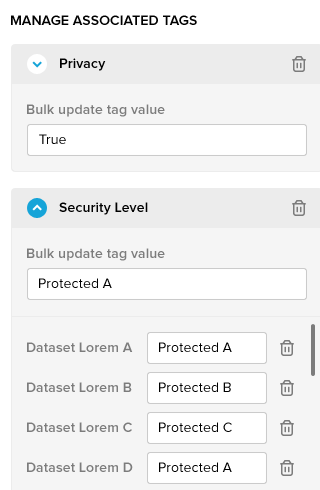
Asset/Dataset Page - Filters



These filters are used on the Asset/Datasets. A call to POST /datasets/search is used to retrieve the filtered results:

* 1. Corresponds to ‘datasetAssetName’.
* 2. Corresponds to ‘tagTypeNameFilterType’.
* 3. Corresponds to ‘tagTypeNameFilterValue’
* 4. Corresponds to ‘createdDateMin’ and ‘createdDateMax’.
* 5. Corresponds to ‘createdBy’.
* 6. Corresponds to ‘lastUpdatedDateMin’ and ‘lastUpdatedDateMax’.
* 7. Corresponds to ‘lastUpdatedBy’.
* 8. Corresponds to ‘serverName’.
* 9. Corresponds to ‘version’.
* 10. Corresponds to ‘sourceSystem’.
* 11. For now, the property is unsupported. (Can be omitted/ignored)
* 12. For now, the property is unsupported. (Can be omitted/ignored)
* 13. For now, the property is unsupported. (Can be omitted/ignored)

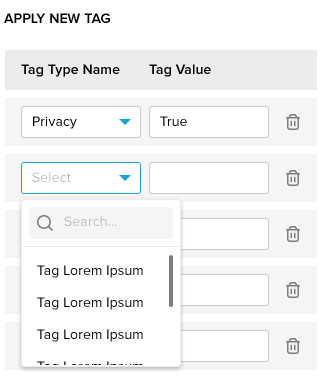
Asset/Dataset Page – Manage Associated Tags



The Associated Tags for a dataset can be managed by calling POST /tags/associations/bulkUpdate. The following section shows how to populate the request body:

* 1. The ‘datasets’ property should contain the ids of all the selected Datasets (with checkbox).
* 2. The selected Datasets already have the tag data present, from the ‘tags’ array for each dataset. This data can be used to populate the Manage Associated Tags form.
* 3. If the trashcan for a tag is clicked, then the ‘shouldRemove’ property for the given tag is set to ‘true’. The frontend should also indicate the tag is flagged for deletion (will be final when ‘save’ is clicked, but can be undone before then).
* 4. If a bulk update tag value is provided, then it is used as the ‘bulkUpdateValue’ for that tag. The individual update boxes for that tag are grayed out in this case (a user cannot provide a bulk update and individual update in the same request).
* 5. The individual tags can be modified via ‘individualUpdateValue’ array:
  + 5a. The datasetId will correspond to the id of each individual dataset.
  + 5b. The tagValue will correspond to the provided value of each individual dataset.
  + 5c. If the trashcan is clicked, then the ‘shouldRemove’ should be set to true. The frontend should also indicate the tag is flagged for deletion (will be final when ‘save’ is clicked, but can be undone before then).
  + 5d. It would be more efficient to keep track of the dirty fields, and only send those fields to the backend.

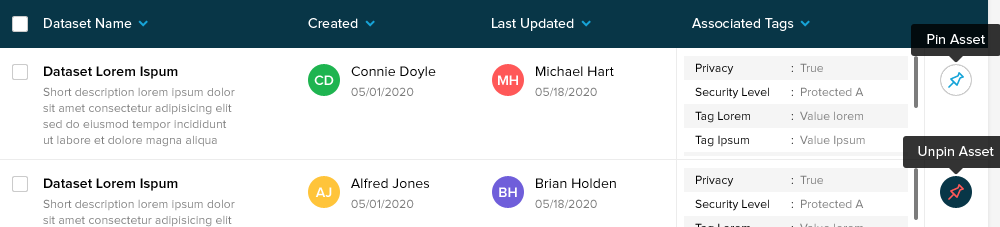
Asset/Dataset Page – Apply New Tag



The Associated Tags for a dataset can be managed by calling POST /tags/associations/bulkUpdate. It is like the Manage Association section:

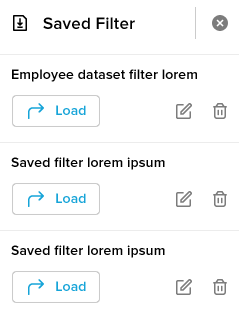
* 1. The ‘datasets’ property should contain the ids of all the selected Datasets (with checkbox).
* 2. The dropdowns can be populated by calling the POST /tags/search endpoint with the ‘tagTypeNameFilterValue’ and ‘tagTypeNameFilterType’ of ‘Contains’.
* 3. The provided TagValue is used as the ‘bulkUpdateValue’.
* 4. ‘individualUpdateValue’ and ‘shouldRemove’ is not needed for this.

Asset/Dataset Page – Pinning and Unpinning Assets



The pinning and unpinning of assets are the responsibility of frontend and stored locally. There is no need to interact with the backend to manage this.

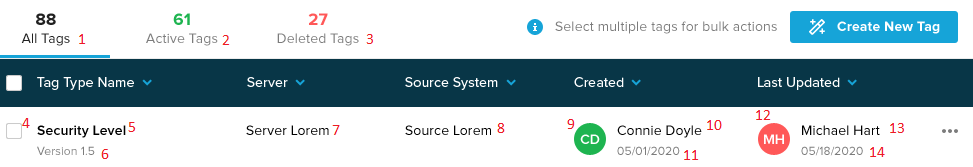
Asset/Dataset Page – Saved Filter



The frontend is responsible for saving and storing the saved filters in local storage.

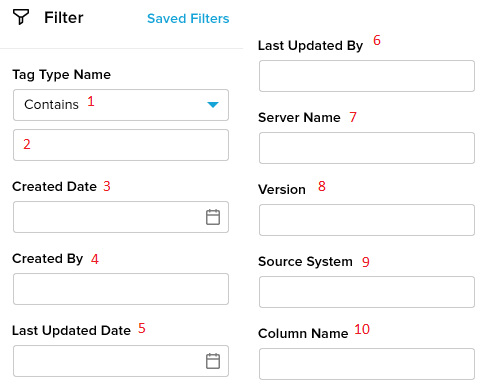
Tags Page – Tag List

The initial list of Tags can be retrieved by providing a blank filter to the POST /tags/search. Additional filtering is possible (see next section). The API response is used to display the list:



* 1. This corresponds to ‘totalTags’ property.
* 2. This corresponds to ‘activeTags’ property.
* 3. This corresponds to ‘totalTags – activeTags’.
* 4. The id is used to identify the data entry for other operations.
* 5. This corresponds to ‘name’ property.
* 6. This corresponds to ‘version’ property.
* 7. This corresponds to ‘server’ property.
* 8. This corresponds to ‘sourceSystem’ property.
* 9. This corresponds to initials of ‘createdBy’ property.
* 10. This corresponds to ‘createdBy’ property.
* 11. This corresponds to ‘createdOn’ property.
* 12. This corresponds to initials of ‘updatedBy’ property.
* 13. This corresponds to ‘updatedBy’ property.
* 14. This corresponds to ‘updatedOn’ property.

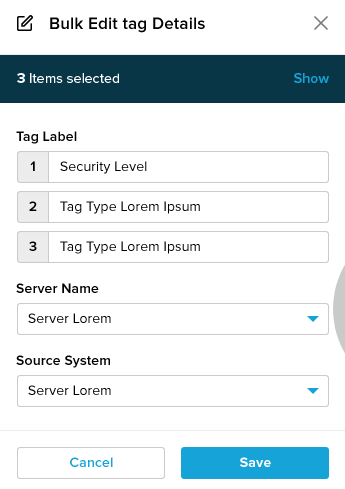
Tags Page - Filters



These filters are used on the Tags. A call to POST /tags/search is used to retrieve the filtered results:

* 1. Corresponds to ‘tagTypeNameFilterType’.
* 2. Corresponds to ‘tagTypeNameFilterValue’
* 3. Corresponds to ‘createdDateMin’ and ‘createdDateMax’.
* 4. Corresponds to ‘createdBy’.
* 5. Corresponds to ‘lastUpdatedDateMin’ and ‘lastUpdatedDateMax’.
* 6. Corresponds to ‘lastUpdatedBy’.
* 7. Corresponds to ‘serverName’.
* 8. Corresponds to ‘version’.
* 9. Corresponds to ‘sourceSystem’.
* 10. For now, the property is unsupported. (Can be omitted/ignored)

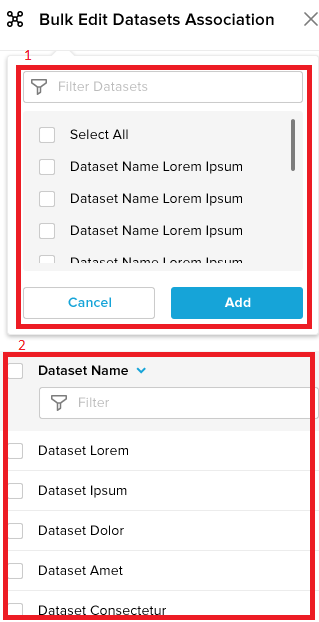
Tags Page – Bulk Edit Tag Details



Bulk edit is done by calling /tags/batchUpdate endpoint. The functionality is as follows:

* 1. Tag Label boxes and id are retrieved from the selected checkboxes in the Tag list.
* 2. The ‘tagLabels’ request body array is populated from the Tag Label form.
* 3. ‘Server Name’ dropdown options are populated from GET ‘/lookups/serverNames’. The selected value is used in the ‘serverName’ request body property.
* 4. ‘Source System’ dropdown options are populated from GET ‘/lookups/sourceSystems’. The selected value is used in the ‘sourceSystem’ request body property.

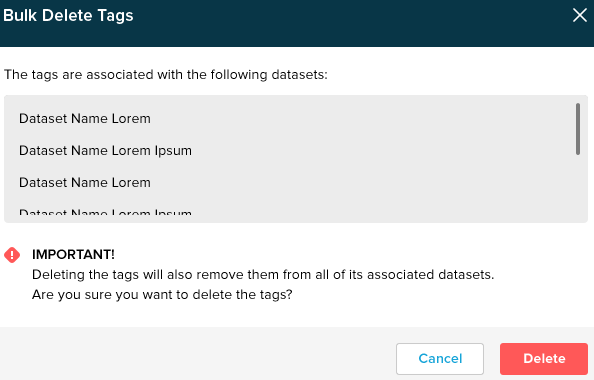
Tags Page – Bulk Edit Datasets Association



Bulk edit of tag-datasets association is done by calling /tags/associations/bulkUpdate endpoint. The functionality is as follows:

* 1. The list of available datasets can be retrieved and filtered by POST /datasets/search with the ‘datasetAssetName’.
* 2. This list is initially populated using the INTERSECTION of the associated Datasets of all the selected Tags.
* For the request body:
  + The ‘tags’ attribute is populated by the ids of all the selected tags with checked checkbox.
  + The ‘editType’ will depend on whether the ‘Save’ or ‘Delete’ button is clicked. ADD is used for ‘Save’ and REMOVE is used for ‘Delete’.
  + datasetAssociations array will contain the selected datasets for ‘Delete’ and will contain all the datasets (in the #2 box) for ‘Save’.

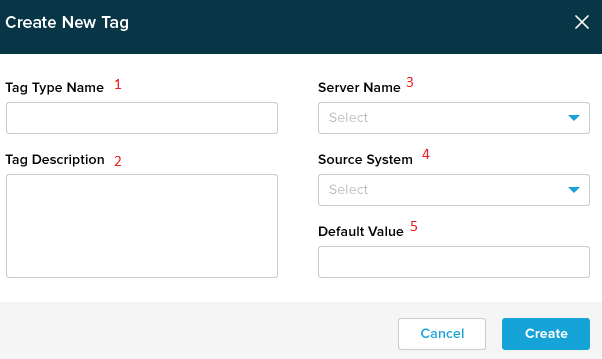
Tags Page – Bulk Delete



Bulk delete of tags is done by calling /tags/batchDelete endpoint. The functionality is as follows:

* 1. An array of ids is provided, containing the selected Tags with checked checkbox.
* 2. The names used for confirmation should already be present in the frontend.

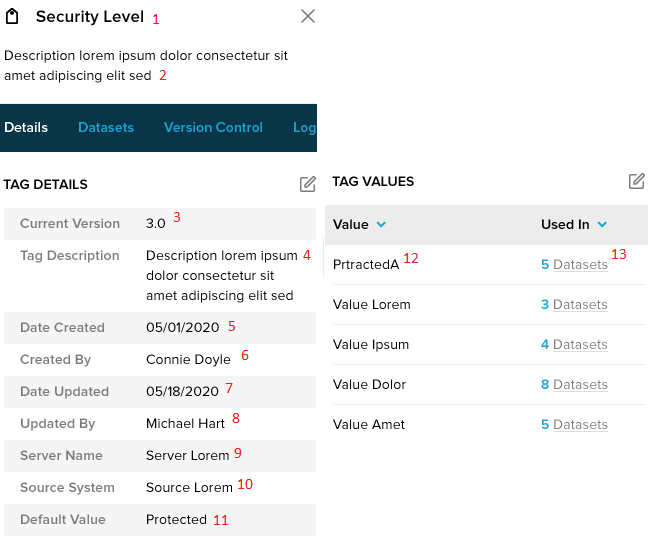
Tags Page – Create New Tag



Create a new tag is done by calling POST /tags:

* 1. Corresponds to ‘name’ in request body.
* 2. Corresponds to ‘description’ in request body.
* 3. Corresponds to ‘server’ in request body. The dropdown can be populated from GET /lookups/serverNames
* 4. Corresponds to ‘sourceSystem’ in request body. The dropdown can be populated from GET /lookups/sourceSystems
* 5. Corresponds to ‘defaultvalue’ in request body.

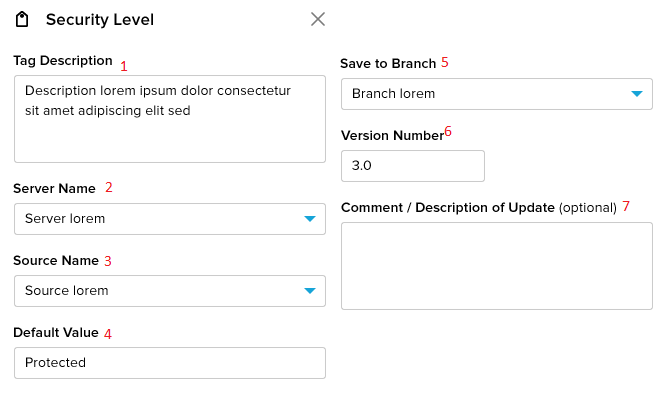
Tags Page – View Tag Details



A detailed view of the new tag can be retrieved by calling GET /tags/{tagId}:

* 1. Corresponds to ‘name’ in response.
* 2. Corresponds to ‘description’ in response.
* 3. Corresponds to ‘version’ in response
* 4. Corresponds to ‘description’ in response
* 5. Corresponds to ‘createdOn’ in response.
* 6. Corresponds to ‘createdBy’ in response.
* 7. Corresponds to ‘updatedOn’ in response.
* 8. Corresponds to ‘updatedBy’ in response.
* 9. Corresponds to ‘server’ in response.
* 10. Corresponds to ‘sourceSystem’ in response.
* 11. Corresponds to ‘defaultValue’ in response.
* 12. Corresponds to ‘tagValues[n].tagValue
* 13. Corresponds to datasets array. On mouseover, the name of the dataset should be shown, and the id can be used to link to the dataset. The size of the array will determine the ‘Used In’ count.

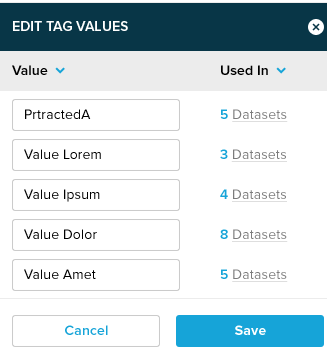
Tags Page – Edit Tag Details



The tag can be edited by calling POST /tags/{tagId}:

* 1. Corresponds to ‘description’ in request body.
* 2. Corresponds to ‘server’ in request body. The dropdown can be populated from GET /lookups/serverNames
* 3. Corresponds to ‘sourceSystem’ in request body. The dropdown can be populated from GET /lookups/sourceSystems
* 4. Corresponds to ‘defaultValue’ in request body.
* 5. This is unsupported and can be ignored.
* 6. Corresponds to ‘versionNumber’ in request body.
* 7. Corresponds to ‘comment’ in request body.

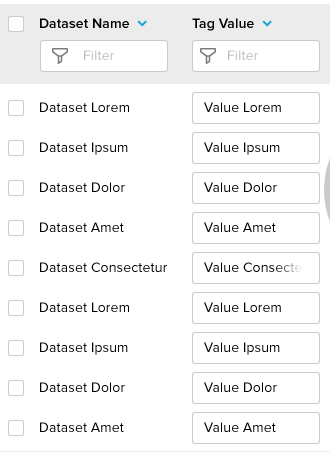
Tags Page – Edit Tag Values



The tag values can be edited by calling POST /tags/{tagId}/updateValues:

* The initial list is already present from the View Tag Details section.
* The frontend should keep track of existing values and also mark which fields are dirty.
* When ‘Save’ is called, an array of ValueUpdateEntry objects are provided in the request body, with the oldValue containing the original tag value, and the newValue containing the updated tag value.
* Only the dirty fields will need to be sent to backend.

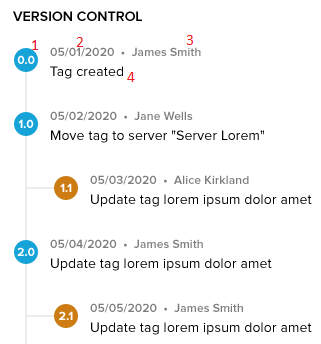
Tags Page – Datasets Association



The tag values can be edited by calling POST /datasets/associations/bulkUpdate:

* The request body ‘datasets’ will contain all the updated datasets.
* The request body ‘tags’ will contain a single element with the current tagId.
* The tags[0].individualUpdateValue will contain multiple elements for each dataset to update. The id of the dataset and corresponding tagValue will be provided.
* If associations are removed, then set the ‘shouldRemove’ flag to be true.

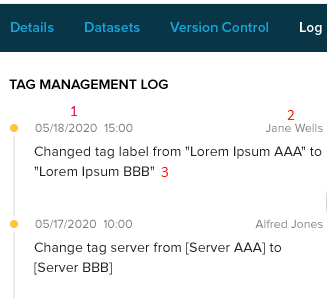
Tags Page – Version Control



The version information can be retrieved by calling GET /tags/{tagId}/versionHistory:

* 1. Corresponds to the ‘version’ in response body.
* 2. Corresponds to the ‘data’ in the response body.
* 3. Corresponds to the ‘actor’ in the response body.
* 4. Corresponds to the ‘comment’ in the response body.
* The frontend is responsible for displaying the tree structure for minor version changes.

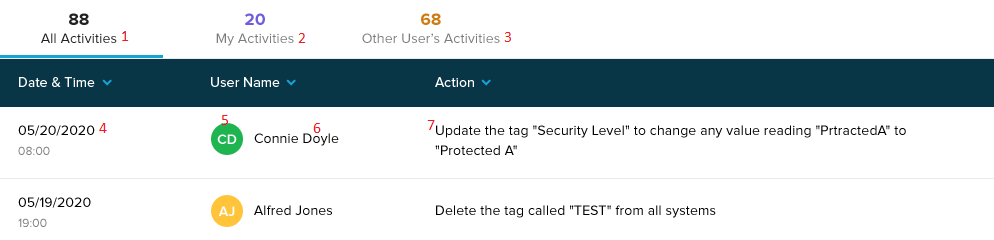
Tags Page – Tag Management Log



The tag management log can be retrieved by calling GET /tags/{tagId}/managementLog:

* 1. Corresponds to the ‘date’ in response body.
* 2. Corresponds to the ‘actor’ in the response body.
* 3. This is a string generated from the ‘type’, ‘propertyName’, ‘oldValue’, ‘newValue’ and ‘datasets’ properties.

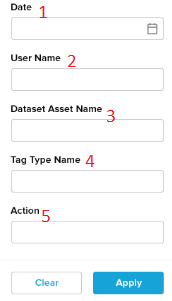
Activities Page – View Activity



The activity log can be retrieved by calling GET /activities:

* 1. Corresponds to ‘allActivities’ in response body.
* 2. Corresponds to ‘myActivities’ in response body.
* 3. Corresponds to ‘allActivities’ - ‘myActivities’.
* 4. Corresponds to the ‘entries[x].date’ in response body.
* 5. Corresponds to the initials of ‘entries[x].actor’ in response body.
* 6. Corresponds to ‘entries[x].actor’ in response body.
* 7. This is a string that is generated from the other properties in entries[x] element, depending on the entries[x].type.
* To generate the action string, a string template should be present for each type of Activity Log. The string template will then use the properties of the entry element to generate the action string. Example: for PropertyChange log entry type, we could use the String `The tag ${tagName} had the property ${propertyName} changed from ${oldValue} to ${newValue}.

Activities Page – Filter Activity



The activity log can be filtered by calling GET /activities with the correct parameters:

* 1. Corresponds to ‘dateMin’ and ‘dateMax’ parameters.
* 2. Corresponds to ‘userName’ parameter.
* 3. Corresponds to ‘datasetName’ parameter.
* 4. Corresponds to ‘tagName’ parameter.
* 5. Corresponds to ‘action’ parameter. It should be changed to be a dropbox, with the choices as per the action parameter enum.

Search Results Page

The search results page is like the Tags Page – View Tags and Dataset Page – View Datasets. The corresponding sections should be sufficient for this page as well.