Daminion and Lightroom tagging integration (an example of using facial recognition in Lightroom)

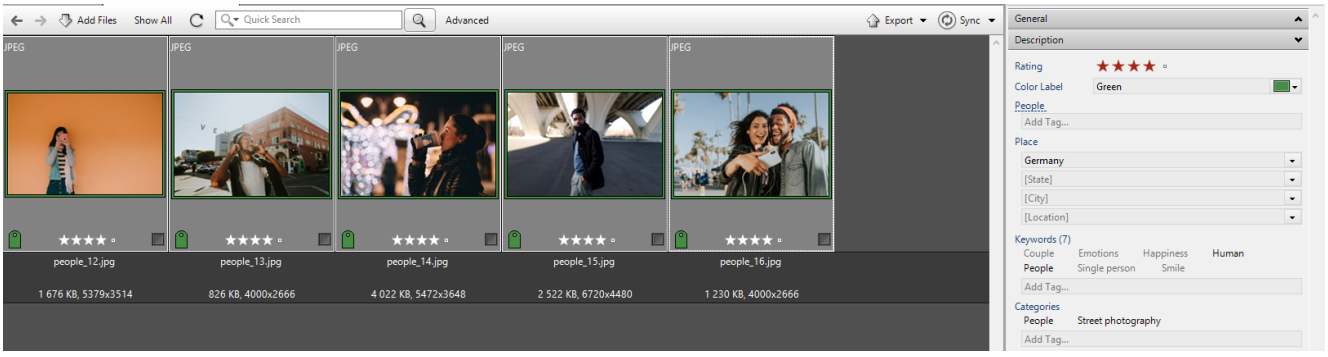
There are two ways of displaying Lightroom's people tags in Daminion:

A – Tag your archive in Lightroom first (facial recognition and\or other tags), save the tags to file's metadata and import your files into Daminion. During the import process, Daminion will automatically pick up all tags from files' metadata and display them in the catalog.

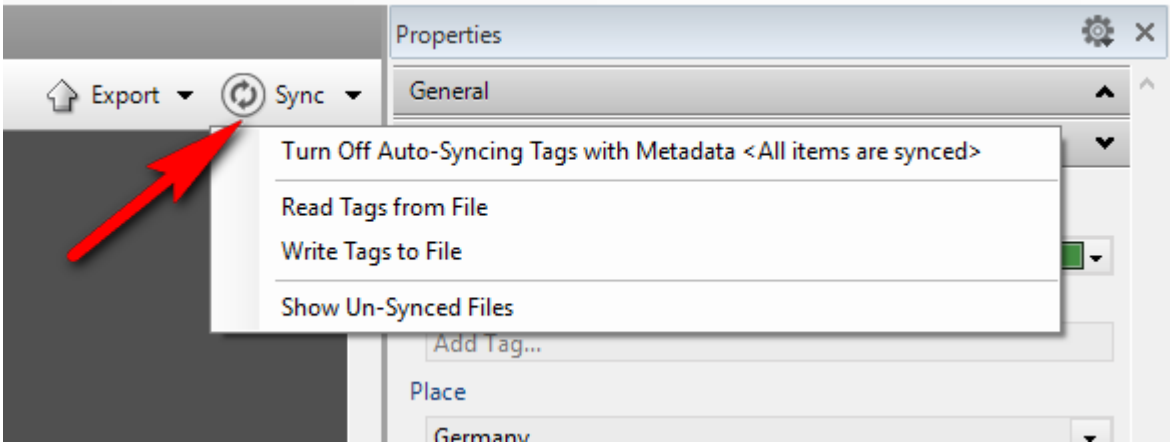
B – Simultaneous work with Daminion and Lightroom

If you have already imported your files into Daminion, and you want to tag people (or assign any other tags) in Lightroom, please follow the steps below:

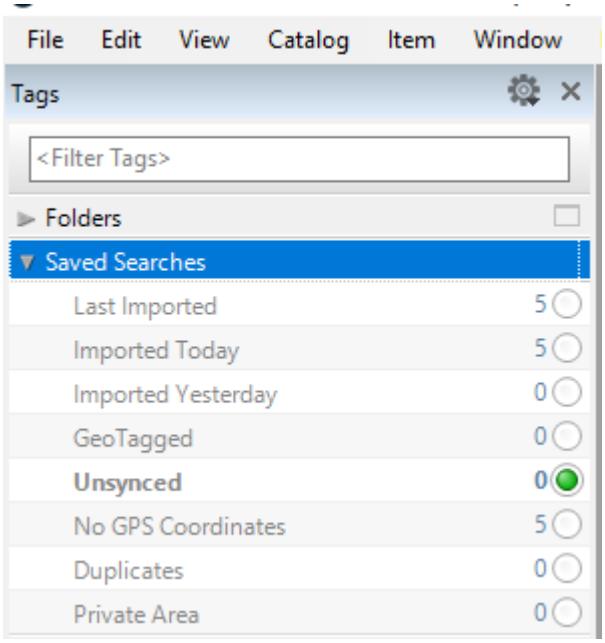
1 – In the screenshot below you can see 5 pictures with several filled-in tags and empty People tags (Daminion).



2 – Before switching to Lightroom and proceeding with tagging, please turn on auto-synchronization in Daminion to make sure that Daminion tags will be written to files' metadata. It will help you avoid tags overwriting.



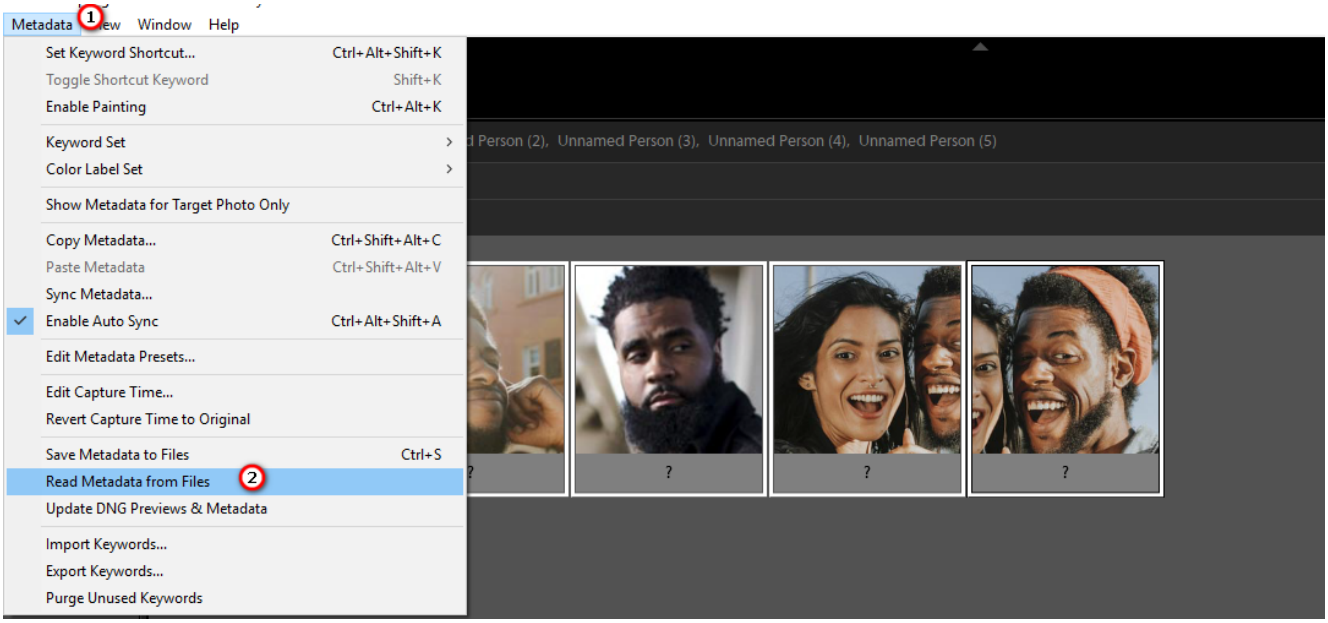
Write your Daminion tags to the files metadata. You can check the number of unsynced files under **Saved Searches** > Unsynced in the Tags tree.



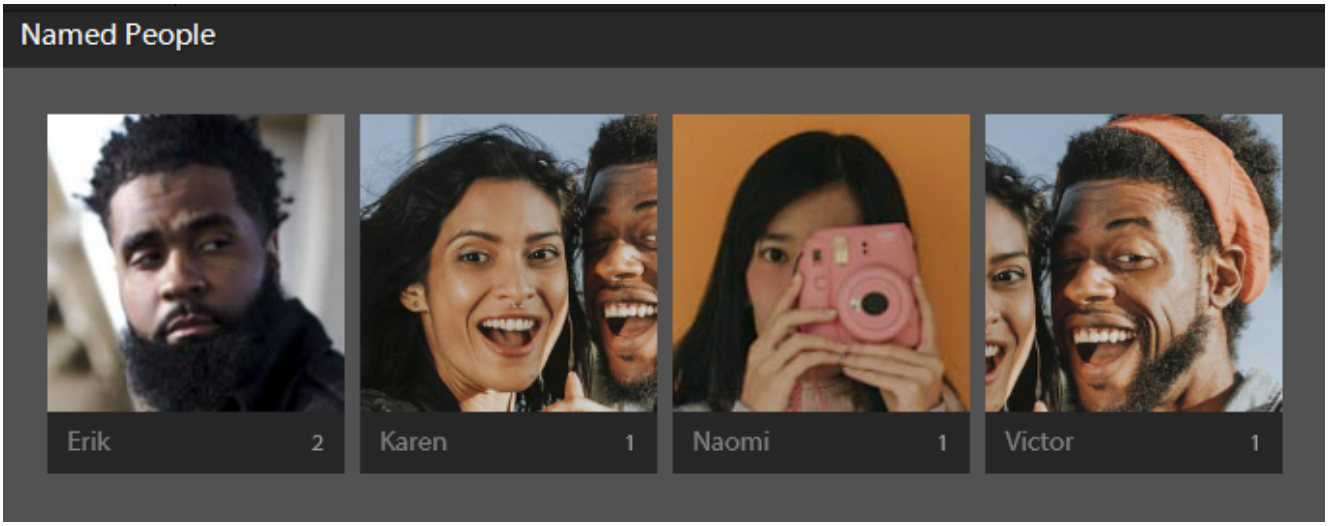
3 – Open Lightroom and make sure that for the selected images the same set of tags is shown as in Daminion.

If not, **please read the tags from files first!**

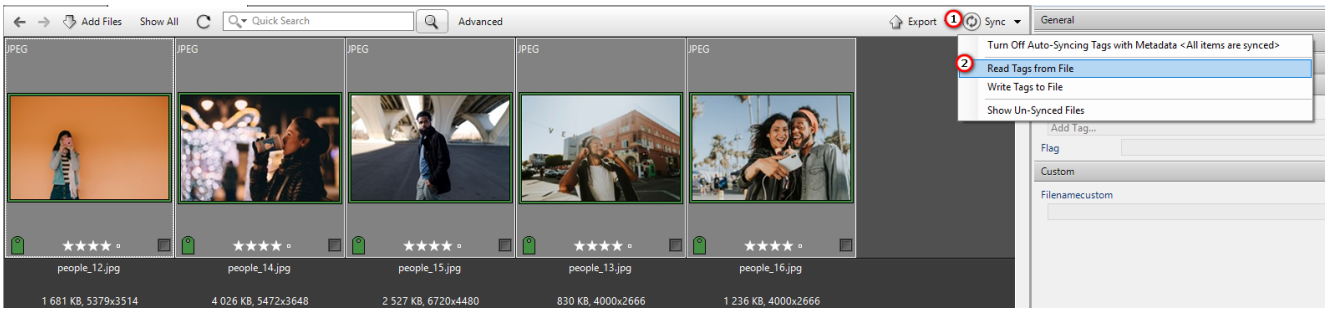
If Lightroom does not display the tags assigned in Daminion before you proceed with tagging in Lightroom and then you will write the new Lightroom tags to files' metadata, the tags previously assigned in Daminion will be overwritten!



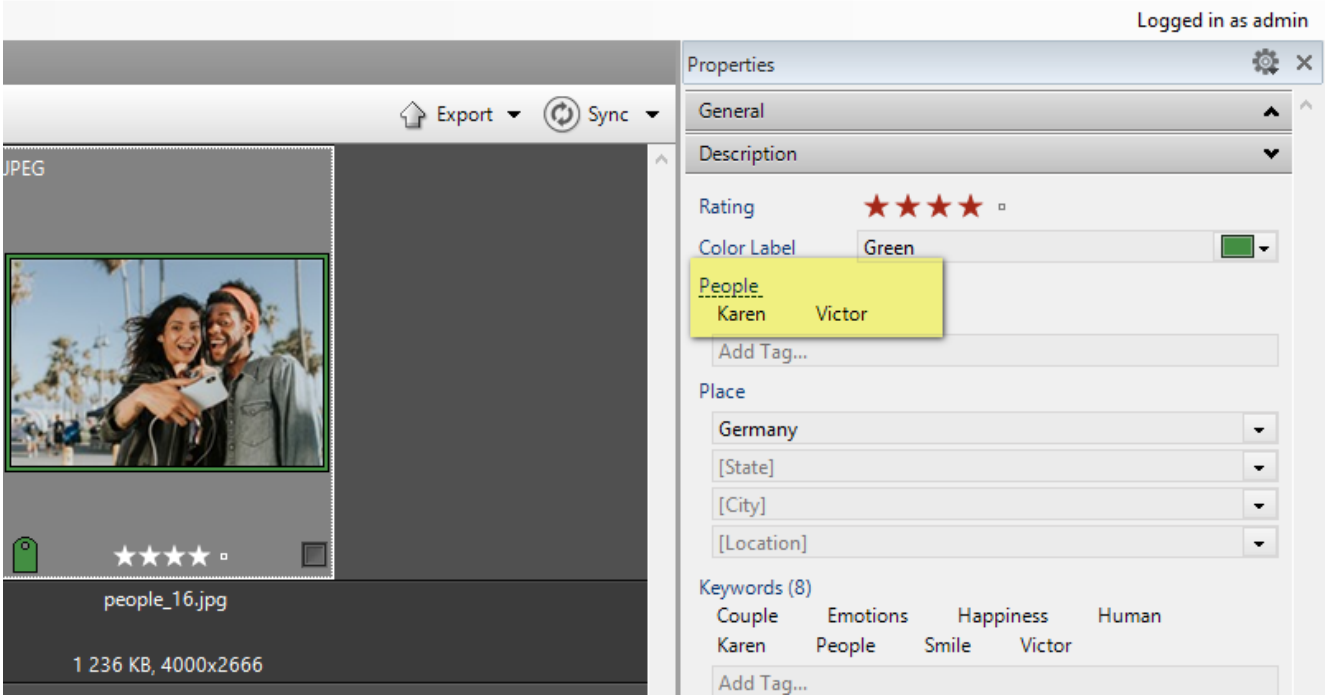
4 – Tag people in Lightroom and save tags to file's metadata via Metadata > Save Metadata to Files.



5 – Now in Daminion select the files you edited in Lightroom, click the Sync button and select "Read tags from files".



Daminion will display assigned People tags:



6 – To automatically display metadata changes in Daminion without having to read tags from files, please activate Auto-Rescan.

Conclusion: To ensure that working with Daminion and Lightroom will be smooth and without data loss, verify the metadata in Lightroom before assigning new tags. If the metadata in Daminion differs from Lightroom, please read the tags from files in Lightroom first!

*Note: before running any test on your database, make sure to create a copy of your local *.dmc catalog or a backup of your shared catalog via Daminion server admin panel > Catalogs > Select a catalog > Actions > Backup*

REST API Interface

Daminion Server Rest API (0.9 Beta)

The REST API lets you integrate external applications with Daminion Server using simple HTTP methods. You can easily fetch various piece of information including: a list of media items, media item properties (tags), thumbnails and previews, list of tags, etc…

By default REST/API is switched off. To enable it you need to open the Daminion Server administration panel, go to the Administration > Catalogs. Right click on a catalog and navigate to Edit. Switch to the API tab and check the “Enable REST API” option.

Base Url

Base url that you need to use in all queries is: **http://hostname:8090/api/**

Where 8090 is a port that can be changed from the Daminion Server Administration Panel (File > Preferences > API)

GET /tags/savedSearches

Gets list of the saved searches

Url Structure:

/api/tags/savedSearches

Example of Usage:

http://localhost:8090/api/tags/savedSearches

Response:

```
<tagValues count="N">
<tagValue id="%value1%" name="%name1%"/>
<tagValue id="%value2%" name="%name2%"/>
…
<tagValue id="%valueN%" name="%nameN%"/>
</tagValues>
```

GET mediaitems/?savedSearchesId=

List media items imported into Daminion Server catalog according to a specified Saved Search.

Url Structure:

/api/mediaitems/?savedSearchesId=%savedSearchId%&offset=%offset%&limit=%limit%&[tags=%tags%]

Mandatory Query Parameters:

- **savedSearchId:** saved search id (see GET /api/tags/savedSearches)
- **offset:** 0-based starting index for the list of results.
- **limit:** maximum number of results to return.

Optional Query Parameters:

tags: what tags should be included into the response. Possible values are: none, all. Default is all.

Example of Usage:

http://localhost:8090/api/mediaitems?savedSearchesId=1&offset=

0&limit=25

Response:

```
<mediaItems count="N">
<mediaItem id="%id%"
uri="http://%host%:%port%/api/mediaitems/%id%">
<tagValues>
<tagValue key="%tagGuid%" name="%tagName%">%value%</tagValue>
<tagValue key="%tagGuid2%"
name="%tagName2%">%value2%</tagValue>
…
<tagValue key="%tagGuidN%"
name="%tagNameN%">%valueN%</tagValue>
</tagValues>

</mediaItem>…
</mediaItems >
```

GET mediaitems/mediaItemId=

Lists tag values of a specified media item.

Url Structure:

/api/mediaitems/%mediaItemId%

Mandatory Query Parameters:

mediaItemId: media item id

Example of Usage:

http://localhost:8090/api/mediaitems/mediaItemId=%itemId%

Response:

```
<mediaItem                                          id=”%id%”
uri=”http://%host%:%port%/api/mediaitems/%id%”>
<tagValues>
<tagValue key=”%tagGuid%” name=”%tagName%”>%value%</tagValue>
<tagValue                                  key=”%tagGuid2%”
name=”%tagName2%”>%value2%</tagValue>
…
<tagValue                                  key=”%tagGuidN%”
name=”%tagNameN%”>%valueN%</tagValue>
</tagValues>
</mediaItem>
```

GET mediaitems/%itemId%/thumbnail?=

Gets thumbnail of a specified media item

Url Structure:

/api/mediaitems/%itemId%/thumbnail?size=%thumbnailSize%

Mandatory Query Parameters:

itemId:media item id

Optional Query Parameters:

size:size of the output thumbnail. Possible values are: small, medium, large. Default is **medium**.

Example of Usage:

http://localhost:8090/api/mediaitems/75/thumbnail?size=large

Response:

Returns “image/jpeg” thumbnail.

GET /mediaitems/%itemId%/preview

Gets preview of a specified media item

Url Structure:

/api/mediaitems/%itemId%/preview

Mandatory Query Parameters:

itemId: media item id

Example of Usage:

http://localhost:8090/api/mediaitems/75/preview

Response:

Returns “image/jpeg” preview

GET /tags

Gets list of the tag specifications.

Url Structure:

/api/tags

Example of Usage:

http://localhost:8090/api/tags

Response:

```
<tags count=”N”>
<tag key=”%tagGuid1%” name=”%tagName1%”/>
<tag key=”%tagGuid2%” name=”%tagName2%”/>
…
<tag key=”%tagGuidN%” name=”%tagNameN%”/>
</tags>
```
