Stagsi Cookbook

This comprehensive document is a hacker's guide to *Stagsi* – an *Interface to Soletude's Tagging System*.

With *Stagsi* you and your team can organize data with a hierarchy of tags. While this is not new (see Alternatives), the openness and flexibility of the data model, QueryLanguage, applicability to any file format, and durability (fault tolerance and number of managed objects) are distinguishing *Stagsi* even from the industry leaders (*Adobe Bridge* and *Apple iTunes*).

Stagsi is a commertial product but its *Free* edition can be used in non-commercial environments with a few light limitations, and a *Professional* license costs pennies compared to others and is irrevocable and non-expiring by design.

- *Stagsi* homepage: go.soletude.ca/stagsi.
- Community forum: go.soletude.ca/stagsi/forum.
- Contribute your scripts: go.soletude.ca/stagsi/recipe.
- Up-to-date online documentation: go.soletude.ca/stagsi/cookbook.
- Offline documentation in various formats: go.soletude.ca/stagsi/archive/cook….

Data File Structure

Stagsi is using simple, easily accessible data storage mechanisms. At its core lies `Stagsi.sqlite` – a standard SQLite version 3 database which is described in detail in its own section.

Other files and folders that may appear:

- `Temp` directory – holds temporary data during import, search and other activities.
- `Icons` directory – holds icons assigned to tags, displayed on top of the objects tagged with them. First, `TagRowId.png` is searched; if not found and the tag is *system* then `System.png` is also searched (e.g. `_mp3.png`).
- `Attachments` directory – holds extra files and folders connected with specific objects.
- `Sorting` directory – holds persistent data used by SortingPlugins.
- `Tools` directory – these files are directly accessible from the main menu. Normally you put useful scripts here, e.g. to back up the database or clean `Temp`. See Tools.
- Numbered directories – hold imported object thumbnails and data, e.g. images, documents, etc. or links to them (for objects imported in linked mode).
- `JSON` files – hold program settings and state (window positions, list of tabs, etc.).
- `Welcome.html` – a web page displayed when opening the database in interactive mode (not with maintenance operations like `/rehash`). See Welcome.

Subfolders in this, `Temp`, `Attachments` and other subdirectories are used to work around certain file systems (especially FAT) limitations on big number of files that become increasingly slow when a directory contains more than a few thousand of entries. *Stagsi* is designed to work with hundreds of thousands of objects (each object having multiple files).

If you wish to store custom data within a *Stagsi* database directory, prefix your (root) files and folder with an underscore: `_Docs\License.txt` or just `_License.txt` if it's directly under the data path. Such names are guaranteed to never conflict with *Stagsi*'s own data.

Welcome Page

If the database folder contains `Welcome.html` file, it's presented to the user whenever it's opened in *Stagsi*. Because it's a local file, you can use the `file://` scheme and reference resources (`<img>` and others) relatively to the database folder. However, interaction with remote resources and `<script>` (JavaScript) are subject to standard *Internet Explorer* and Windows restrictions. Basic formatting with local images and links (`<a>`) should always work but remember that *Internet Explorer* is fairly old and doesn't support many modern web standards.

If you're redistributing your database, you can put a license text here, or information about and links to your company or team.

You cannot directly launch processes from this page but you can reference executable files like `<a href="file://run_me.bat">` and when the user clicks on such a link he'll be asked if he wants to run it. This can be used to open tabs in *Stagsi* (`Stagsi.exe /search "the | query"`), run maintenance tasks (`/rethumb`), run arbitrary programs (`Notepad.exe`), etc.

Skeleton

`Skeleton` appears under *Stagsi*'s installation directory and holds initialization files for new databases (done when "opening" an empty directory by copying `Skeleton` files there). It is used to specify new database defaults.

If `Skeleton\Temp\Bootstrap.bat` exists then it's ran within CommonScriptEnv in the end of initialization; *Stagsi* waits until it exits (ignoring its exit code) and proceeds on loading the new database. To cancel database creation use `taskkill /F /PID %pid%`.

Example of things you can do thanks to this folder:

- Set configuration – e.g. UI `Language` in `Settings.json`.
- Assign tag icons – usually referenced by the tag's *system* name because `RowId` may not be known at that time – `Icons\_Deleted.png`.
- Send a notification to a web page from `Temp\Bootstrap.bat`.
- Add useful Tools or Plugins.
- Check-in the new database into a version-control system such as with `git init`.
- Change tag colors in `Stagsi.sqlite` – `Color` field of the `Tags` table.
- Add custom SQLite columns or rows – e.g. `DataLicense` `Key` in `Stags` (displayed in the database properties in *Stagsi*).
- Add commonly used `Tags` or group all system tags into one parent to make it less messy using `ParentRowId` of the `Tags` table.

Temp

May include such files, grouped into arbitrary subfolders:

- `DLL` files – compiled search queries. *Stagsi* keeps number of these under some limit, removing old searches automatically.

  Stagsi is a Net program, and when you perform a search – the search expression is compiled as a Net assembly stored in `Temp` for better performance (even when *Stagsi* is restarted). The search will still work without write access to `Temp` (such as in `/ro` – read-only mode) but even identical searches will be compiled from scratch which takes a few hundreds of milliseconds.

- `SQLITE` files – old database versions. As you upgrade *Stagsi*, sometimes it will involve upgrading the database file. This is an irreversible process (older *Stagsi* version will no more work with an upgraded database). Before upgrading, the database is copied to `Temp` so that the upgrade can be manually reversed (of course, losing changes done after the upgrade). You shouldn't see too many of such files, but in any case are not used by *Stagsi* and can be safely deleted.
- Other files – created during import for holding file data and thumbnails. Normally these are removed automatically unless something went horribly wrong (*Stagsi* crashed, the PC halt and caught fire, aliens invaded Windows, etc.).

  If a *Stagsi* process has created these files, and then exited – these files are never going to be used again.

If *Stagsi* is not running, it's safe to remove this entire folder or any files inside it to reclaim space.

Numbered Directories

Each numbered directory includes one or more groups of files belonging to one particular imported object (e.g. an image, a video or a document). The directory in which a group is stored is determined by

the object's `RowId`.

Note: `RowId` is unique. *Stagsi* (or, rather, its SQLite backend) guarantees that once a `RowId` was allocated (by importing a new object), it will never be reused, not even if the object is deleted and another is imported (a gap will be present, where the old object's `RowId` is left unused forever within this database).

Let's assume that:

- `RowId` of the imported object is `12345`
- Its format is `MP3`
- Your thumbnail format is `PNG`
- The number of groups per directory is 1000 (the default)

In this case, all files of this group (of this imported object) are stored in the directory named `12` – the result of rounding down the result of dividing `RowId` by `1000` (the limit).

Such files may exist for this group:

- `123.mp3` – the actual file data. Only if this object was imported in non-linked mode (*Copy* or *Move*).
- `123` – an extensionless file existing for objects imported in linked mode. It's a regular text file that can be opened in *Windows Notepad* or similar program (even though it has no conventional `.txt` extension). Its content is the path to the "real" file somewhere else (to which this object is "linked") – absolute or relative to the database path (folder with *Stagsi.sqlite*), possibly with `%environment%` variables.
- `t123.png` – full-size thumbnail generated from the object's file data.
- `u123.png` – user-supplied full-size thumbnail.
- `c123.png` – cropped thumbnail, either from `u123.png` or, if that doesn't exist, from `t123.png`.

Thumbnail Files

There are 3 possible thumbnail files per each imported object which differ in the prefix (first letter): `t`, `u` and `c`.

`t` stands for s"t"andard thumbnail. *Stagsi* generates this file from the file data for known file formats (like images). For unknown formats, e.g. for MP3 audio or XLS spreadsheet, this file is not created.

`u` stands for "u"ser thumbnail. *Stagsi* doesn't create or change this file except via the *Edit Thumbnail* dialog. It doesn't exist for objects using standard thumbnails (`t`).

`c` stands for "c"ropped thumbnail. This is a smaller version of either `t` or `u`. It's managed by the user via the *Edit Thumbnail* dialog. It doesn't exist for objects using full-size thumbnails (either `t` or `u`).

This table summarizes the differences of each prefix:

Prefix	Can be manually changed?	When used?	Vital for backups?

t	No – change `u` or `c` instead.	If `u` and `c` don't exist.	No, generated from file data with `/rethumb`.
u	Yes, when *Stagsi* is not running. If both `u` and `c` exist, make sure to change both of them accordingly to avoid discrepancies later.	If `c` doesn't exist.	Yes.
c		Always, if exists.	No, generated from `c` or `u` with `/rethumb`.

Relative Links

By default, when files are imported in *Link* mode, they reference original files by the absolute path. This is fine if the database is used on the same system or its location often changes (while location of the original files doesn't). However, it's troublesome if the is being used on different systems (e.g. via a shared folder or *git* repository), or original files are often moved.

Let's imagine you have a folder with books and a *Stagsi* database which provides a "view" into that folder (book files are not inside *Stagsi* but can be searched and opened from there). We need the book folder and the database to be "portable" – usable from a USB stick on different systems (even without *Stagsi* installed), or when shared via *Dropbox* or *GitHub*, etc. which is impossible with absolute paths (the default) since files end up in different locations.

If you are sharing the book files as well as the database, you can place the database in the root folder with the books and change all link data files from `C:\Users\MyBooks\...` to `..\` (so that file data is searched 1 level above *Stagsi.sqlite*):

```
_Stagsi\
Bios\
Computers\
Fantasy\
History\
...
```

If you are sharing the database only and your users have the book files somewhere, you can use what is called "environment variables". These are basically arbitrary strings with a predetermined name, set via *Control Panel* > *System* > *Advanced settings* > *Environment Variables*. Create one named `STAGSI_BOOKS`, set it to `C:\Users\MyBooks` (and ask other users to do the same) and replace link data files from `C:\Users\MyBooks\...` to `%STAGSI_BOOKS%\...`.

Fixing Links

Normal object data is stored in files like `123.gif` but if an object was imported in "linked" mode, the extension is omitted: `123` and the file stores just a plain text absolute path to the "real" object file (outside of the database directory).

To fix a single link, it's usually more convenient to simply re-import the file whose link is broken – *Stagsi* will detect it as a duplicate and do nothing except fixing the link to point to the new location of that file.

To fix links in a more complex way, these files can be edited by hand (if *Stagsi* is not running).
For example, if you have moved a folder to which multiple links exist, you can run a *Find & Replace* on this directory using an editor like *Notepad++*. If the old folder's location was `C:\MyDocuments` and

new location is `F:\My` then here is how it can be fixed:

On a big database, this can take longer than desired because we're matching all files within the database. To speed it up, we can pre-filter files by wildcards. In *Notepad++*, setting the `Filters` input to the following:

```
?;??;???;????;?????;??????
```

...will restrict the search to files which names are exactly 1, 2, 3, 4, 5 or 6 characters long. This still includes some files with extensions like `12.gif` but not more than first 99 `RowId`'s (if using traditional 3-character long extensions), leading to a significant performance benefit.

In a program that allows exclusion by wildcard (like *WinRAR*), specifying `*.*` is better as it skips all files that have an extension regardless of their name's length.

If you a scripting guy, a little bit of *Perl* can make anybody's life easier. Here's code that replaces absolute path references (produced by the import in non-link mode) with relative:

Misc/Unprefix link paths.pl • Skip this example (#exQW) • Browse on GitHub • Adjustments (2)

```
chdir q[C:\My\Database];
@files = glob "*\\*";

foreach my $file (@files) {
  if ($file =~ /\\d+$/) {
    open(my $in, $file) or die $!;
    my $content = <$in>;
    # .\ is superfluous, could be just an empty part.
    $content =~ s/^C:\\My\\Database\\/.\\/;
    open(my $out, '>', $file) or die $!;
    print $out $content;
  }
}
```

Attachments

Stagsi allows adding extra files and folders per each database object. For example, for a video lesson you can keep notes in a simple text file. Such extras are placed in separate directories per object within `Attachments` using the same numbering convention as the main directory. For example, object with `RowId` of 12345 will use `Attachments\12\12345\`.

Object directory contents are entirely user-dictated and may contain any kind of files and even subfolders. *Stagsi* does not track or hash them.

Common Script Environment

When *Stagsi* starts new processes it typically sets their environment as follows:

- Working directory is set to `DataPath` (current database path).
- `STAGSI` environment variable is set to the path of `Stagsi.exe`.
- `PID` environment variable is set to *Stagsi*'s process ID.

Tools

Many *Stagsi*-related tasks can be automated with scripts or small programs. The `Tools` directory and the corresponding main menu command allow calling them easily from within *Stagsi*. You can write your own scripts or find a bunch of helpers created by the community at go.soletude.ca/stagsi/recipe.

If you access a certain tool often, create a hotkey by putting a `&` before a symbol in its file name, for example: `Clean T&emp.bat` – then call it by pressing Alt+T (opens the `Tools` menu) followed by Alt+E (Alt + symbol after the `&`).

`Tools`' processes are started within CommonScriptEnv. For files starting with `Selection` (e.g. `Selection to iTunes.bat`) and when *Object Browser*'s selection is non-empty, an environment variable named `SELECTION` is set to a path to a text file containing information about selected object(s) in the same JSON format as *Object Browser*'s `Copy` command. Tools using this file for long (if another tool may be called before this tool exits) should copy or move this file elsewhere and work on that copy, otherwise it may change while the tool is running.

Scale and Convert Selection to PNG

Suppose you have a database of icons for mobile apps and, naturally, they are all vectors (SVGs). *Stagsi* can display their thumbnails thanks to its plugins but when you want to use such an icon in a program that doesn't support SVG you have to convert it into PNG and doing it by hand is tedious.

Below is a PHP script that's using *ImageMagick* to convert images selected in *Stagsi* into PNGs of certain square dimensions, preserving their transparency and trimming transparent background on the edges (`-trim`). Copy/paste it and change `$size` to create multiple "tools", each converting to different dimensions (e.g. 16x16 and 32x32):

{} Tools/Selection to &16x16 PNG.php • Skip this example (#exVA) • Browse on GitHub • Adjustments (2)

```
<?php
$size = 16;
$imagick = "C:\\Program Files\\ImageMagick\\convert.exe";

strtok(file_get_contents(getenv('SELECTION')), '{');
$data = json_decode('{'.strtok(null));

foreach ($data->Objects as $obj) {
  system(escapeshellarg($imagick).
         " -size {$size}x$size -background none -trim".
         ' '.escapeshellarg($obj->FilePath).' '.
         '
'.escapeshellarg(getenv('USERPROFILE').'\\Desktop\\'.$obj->Title."
{$size}x$size.png"));
}
```

Backup Tools

Nobody wants to invest their time managing thousands of items if there is no clear backup strategy (this is one problem with proprietary software). *Stagsi* is completely open on how it stores the data making any conventional tool perfectly adequate for the task.

Read *What to Backup?* to learn which files you have to store and which can be safely left out.

As for the tools, here are some options:

Utility Class	Tools	Pros	Cons
Archivers	WinRAR, 7-Zip, etc.	Fast (literally 1 click if using WinRAR profiles). Extremely simple to use. Encryption and corruption protection (WinRAR) out of the box.	Subsequent backups are as large as the entire database which is a big problem for frequent backups.
Version Control Systems	SVN, git, Mercurial, etc.	Drastically reduced storage requirements. Some are distributed (can be used as a "shared folder").	Steeper learning curve. Slow commits with tens of thousands of files. Incremental implies slow restoration and larger impact of a history corruption.
Cloud	ownCloud, Dropbox, G.Drive, AWS, etc.	Effortless and simple to use. Distributed and (reputedly) reliable.	May be paid. May be third party. Not versioned – can't revert to an "earlier version" if your folder was synced after corruption.
Backups	dar, tarsnap, Acronis True Image, VSS, etc.	Pros and cons are very specific to a particular product. For instance, cross-platform open-source *dar* allows incremental backups with encryption, volume support, scripting, etc. but it aimed at power users. *Acronis True Image* is the opposite, more like a "backup cloud" for home users.	

What to Backup?

With *Stagsi*, you don't need to back up the entire data directory because some data can be reconstructed from another. However, omitting wrong files may lead to small but unrecoverable database.

Short and safe answer: use a tool like *WinRAR* or *git* that supports excluding files by wildcards and specify these:

```
Temp/*
*/t*.*
*/c*.*
```

In other words, omit the `Temp` directory (DataTemp) and files in the numbered folders starting with `t` or `c`. After extracting a backup made this way, run the `/rethumb` command and you are ready to go.

With *WinRAR*, you can make backups significantly faster if you add the following to the list of files added without compression (since they're already compressed by nature):

```
*.gif *.png *.jpg
```

WinRAR also supports storing settings under profiles – prepare it once and back up with a single click.

Long answer: the following files are of interest, in order of importance:

- `Stagsi.sqlite` – the most critical file. If it's missing, nothing at all can be salvaged but even if you have this file only, you can recover your tag hierarchy, list of imported objects and their tags, locations, hashes and other metadata. No thumbnails and object data will be available but if you manually replace those, you can fully restore the database.
- `Settings.json` – holds the configuration of this database. It's important but, if missing, it can be rewritten by hand with some effort. Best to keep.
- In numbered folders, files with numeric base name, e.g. `123.gif` or `123`. These are data and link files. If missing, commands like `Open` and `Export` won't work (because there's no file to open or export), and commands like `/rethumb`, `/rehash` and `/check` will fail. Best to keep.

  Obviously, if you have linked objects – you should ensure their targets are backed up too.
  For example, if you back up the `123` file but not the `C:\My\Picture.jpg` file – you'll store the link to this file (which is good) but no actual file, so to recover the backup you'll have to find that `Picture.jpg` elsewhere.

- In numbered folders, files with names starting with `u`, e.g. `u123.png`. These are user-assigned thumbnails. If missing, *Stagsi* will use its own thumbnail (`t123.png`) or no (for unknown formats like PDFs). Best to keep.
- `Attachments` directory – extra files and folders like notes and bookmarks that you've created with attachment object menu commands. They're not used at all by the database but if they're important to you, then keep this directory.
- `Tools` directory – helper scripts and programs for calling from within *Stagsi*. If you're using them then you should keep this.
- `Icons` directory – holds tag icons. If missing, tags will lose all icons but this is hardly critical. Still, better to keep.
- Other `*.json` files – hold state of the program, like filled tag decks, sort modes, etc. Not very important, you'll start off with a blank (default) interface if missing, so it's up to you.

And the following files are not important:

- In numbered folders, files with names starting with `t` or `c`, e.g. `t123.png`. These are automatically generated full-size and cropped thumbnails. They can be fully regenerated with `/rethumb` provided other data is safe.
- `Temp` directory – can be entirely omitted. Nothing needs to be done if it's missing. See DataTemp.
- `Sorting` directory – if cleared partially or fully, will be regenerated upon next search sorted by the corresponding SortingPlugins.

SQLite

What is SQLite?

SQLite is a database engine used by world-popular projects like Firefox and Android. *Stagsi* is using SQLite version 3 as a backend for storing metadata (image properties, tag hierarchy, etc.). Advanced users can manipulate this database directly using the standard tools.

SQLite Browser for Windows, Linux and Mac

Homepage: sqlitebrowser.org.

This is a free, easy to use program for all popular operating systems.

Let's imagine you have a large number of objects with a typo in the title: "gmae" instead of "game". You can fix this by issuing this SQL query (note that it's case-sensitive – "Gmae" isn't affected):

```
UPDATE Objects SET Title = Replace(Title, "gmae", "game")
```

Or suppose you want to wrap titles of all tags' of some parent into brackets. For this, first determine the parent tag's `RowId` by hovering over it in *Stagsi*'s tag dialog, then utilize the SQL's concatenation operator (`||`):

```
UPDATE Tags SET Title = '[' || Title || ']' WHERE ParentRowId = 12
```

SQLite Firefox Addons

If you have Firefox installed, then you can avoid installing any other software for simple SQLite tasks.

Usage is similar to *SQLite Browser*. For example, you can multiply the Weight of all objects' tags' by 10 with this SQL query:

```
UPDATE ObjectTags SET Weight = Weight * 10
```

sqlite3 Command-Line Tool

The SQLite project provides a cross-platform console tool for running queries on SQLite databases. It's great for use in shell scripts.

Here's a query that returns a comma-separated list of `RowId`'s of objects that are over 10 MiB in size:

```
SELECT GROUP_CONCAT(RowId) FROM Objects WHERE FileSize > 10485760
```

Reading from PHP

If you are a programmer, you can automate many tasks with PHP (or similar languages). PHP comes with built-in support of SQLite 3 via PDO.

Here is a script to update hashes of the specific objects only (handles linked files; doesn't handle large files due to a different hashing mechanism):

Database/rehash simple.php • Skip this example (#exJV) • Browse on GitHub • Adjustments (3)

```
<?php
$stagsiDataPath = "C:/My/db";
$db = new PDO("sqlite:$stagsiDataPath/Stagsi.sqlite");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$getStmt = $db->prepare("SELECT * FROM Objects WHERE RowId IN (1, 2,
3)");
$getStmt->execute();
$rows = $getStmt->fetchAll(PDO::FETCH_OBJ);

$setStmt = $db->prepare("UPDATE Objects SET Hash = ? WHERE RowId =
?");

foreach ($rows as $row) {
  $base = "$stagsiDataPath/".floor($row->RowId / 1000)."/$row-
>RowId";
  $file = is_file($base) ? file_get_contents($base) : "$base.$row-
>Format";

  $setStmt->bindValue(1, md5_file($file));
  $setStmt->bindValue(2, $row->RowId);
  $setStmt->execute();
}
```

Comparing the NULLs

Most people unfamiliar with SQL make a common mistake of assuming that `WHERE Column = NULL` will match if `Column` is `NULL`. However, in SQL nothing matches a `NULL`, not even another `NULL` so `WHERE NULL = NULL` will never match. Instead, for comparing with `NULL` use the `IS` operator: `WHERE Column IS NULL` or `Column IS NOT NULL` (not <>).

For example, to convert names of all your tags to lower case except for system tags like `Last Imported`:

```
UPDATE Tags SET Title = LOWER(Title) WHERE System IS NOT NULL
```

Note: `LOWER()` only works with Latin symbols and will corrupt strings containing other characters.

Tags

System Tags

System tags are special kind of tags maintained by *Stagsi*. Unlike normal tags, they:

- Cannot be created or deleted.
- Cannot be assigned/removed to/from objects.
- Cannot have their Weight changed (it's always 0).

However, just like normal tags, they:

- Can have their title, color, order, parent, icon changed.
- Can be searched for (be part of a search expression).

The following system tags exist:

- `Untagged` – appears on objects which have no normal tags (added by user).
- `Corrupted` – appears on objects which have failed integrity (hash) check. This means their data on disk is different from what it was when the object was imported. For linked objects, this might mean that the object was moved away from its target location.
- Format tags like `GIF` and `MP3` – appear on objects of this format, e.g. MP3 audio tracks.
- `Imported` – when *Stagsi* imports tags from metadata (created by *Windows Explorer*, *Adobe Bridge* or other tools), it creates them under this system tag. Objects don't have this tag.
- `Animated` – appears on objects which can be played within the *Stagsi* itself (normally by hovering). *Stagsi* understands very few formats (most notably – animated GIFs); absence of this tag doesn't mean the object is not playable in an external media player.
- `Last imported` – appears on objects added (or skipped due to duplicates) during the most recent import session.
- `Deleted` – appears on objects in the "recycle bin", i.e. hidden from regular searches but still able to be recovered (by searching for this tag explicitly).

  Because such objects are hidden by default you can use it as a "blacklist" for files that should be ignored on import – e.g. if you have "loosely" duplicate files (e.g. different only in meta-data or a few pixels) in various places that you have once identified and do not want them to clutter your database anymore. Another way would be to create a separate tag like "Blacklisted" and always launch *Stagsi* as `Stagsi.exe /pick /force -Blacklisted` (see pick) but then you won't be able to browse such objects even by searching for this tag explicitly.

- `Custom thumbnail` – appears on objects with user-assigned thumbnails (the `u` file prefix). It's useful if you have imported a bunch of formats that *Stagsi* cannot generate thumbnails for and started to gradually add thumbnails by hand. Searching for `-"Custom thumbnail"` lets you resume with objects yet missing a thumbnail.

Automatic Localization

Actual titles for system tags having blank `Title` in the database are taken from the current localization – this way you and that guy in Russia see the `Deleted` tag differently even if sharing the same database. However, because the localized title is used in the search, queries now become localization-specific and on your system `Удалённые` will give an error.

If this is a problem for you – give system tag(s) an explicit name (even if it matches your localization exactly, e.g. "Deleted"). `Title` will stop being an empty string and *Stagsi* will treat it literally.

Volume Indicator

In most cases, when a tag's name is displayed it will be accompanied by a "volume indicator" – a small vertical "progress bar" indicating how many objects in a database are tagged with this tag using a

logarithmic scale.

For example, if your database has 20 objects in total and 5 of them are tagged with tag `A`, then `A`'s volume is *log10(5+1) / log10(20+1)* = 58.8%.

Weight & Relevance

An object has not just tags but also relevance of that tag to this particular object – its "weight".

Imagine you have 2 images: a portrait of someone and a room with a hanging portrait on the wall. In the first case, the "portrait" tag is much stronger, more relevant to the description of the image than in the second, where it's a mere detail, not the main theme.

Yet in most traditional tagging systems, you get a binary "have tag"/"have no tag" relationships and you have to skip describing small details or use different tags for them (e.g. "portrait_image" and "hanging_portrait") to avoid polluting the search with irrelevant results. Or (as in *Adobe Bridge*) you get a 5-star rating system but it's global and doesn't tell us if the picture is "generally great" (5 stars) or the "portrait frame is excellent" (same 5 stars).

In *Stagsi*, you can still use one tag for both images, but assign +1 weight to the first (a portrait) and 0 to the second (a handing picture). And the equivalent of "generally great" (a generic rating) is just a weight of some specific tag, like "greatness".

Weights:

- Can be arbitrarily large (up to some sane limit like `2147483648`), as well as negative.
- Have 0 as the default value.
- Affect only sorting of the results. No matter the weight, if a tag is assigned to an object and that tag is listed in the search expression – the object will always appear in the results. If its relevance (combined weights) is very low, it may go last, but it will appear nevertheless.
- Relevance sorting is enabled by default and works like this: for each matched object, take all tags that are mentioned in the search expression (except `-negative`), sum up their weights and use that number to compare with other objects (object with higher weight goes first).

  Using our portrait example, given images A (`portrait` weight +1) and B (`portrait` weight 0) and search query of `portrait`, A is put first because its relevance (sum of weights) is 1 while B's relevance is 0.

Filtering the Tree

If you have over a hundred of tags you will find that the *Quick filter* input is the most convenient way to browse your tag hierarchy. Here are some helpful tips regarding both the tags' and saved searches' trees:

- When you first filter the tree *Stagsi* remembers previously expanded nodes. When the filter is cleared, the state of before the first filter is restored (expanded nodes are made expanded, collapsed nodes are made collapsed). In addition, selected nodes are always expanded (see below).
- Filter changes do not affect the checkboxes – even if a node becomes hidden it still remains checked. If you need to check a number of disparate tags – filter the tree several times, each time

checking new nodes to add them to the selection (even though some of the previously checked nodes would not be visible).

- After a filter change, newly hidden nodes are un-selected if they were previously selected. If any node remains selected then after the filter is applied (nodes hidden or shown) the tree expands selected node(s) and scrolls to the first selected node.

  Use this to display a tag within its context: filter a tree, select your tag(s) and clear the filter to reveal its siblings and other tags.

Tag Decks & Deck Sets

One unique feature of *Stagsi* (among many others) is a large array of tagging hotkeys – over 2 dozens.

Hotkeys are bound to tag slots, which can be of two kinds: global and deck-specific. Each slot may be assigned one or more tags that are applied to objects in the current selection or under the mouse cursor (if you've enabled this option).

First, you have *5 global tag slots* that respond to `Q` `W` `E` `R` `T` keys (without modifiers). Using `Shift` modifier would toggle tags instead of adding only. Another `5 history slots` (`Y` `U` `I` `O` `P`) are like global but get automatically filled as you pick tags elsewhere.

Second, you have *10 deck-specific tag slots* that respond to numeric keys (`1` `2` `3` … `0`), again without modifiers or with `Shift`.

Then you have *10 decks* that are switched with `Alt+numeric_key`. So you have about 100 tag slots (each with multiple tags!) that are reachable by at most 2 keystrokes (switch deck, then apply tags).

Finally, if that was not enough, you can *switch decks themselves* by means of a button with a menu (no hotkey this time). The menu allows you to create as many tag decks (organized into *deck sets*) as you wish. So, with 2 clicks and 2 keystrokes at most you can reach pretty much every set of tags imaginable.

Now, a practical example.

Global slots are independent of decks, and they're also few. Fill them with very common, generic tags that you'll likely have, such as: "tag_me" (for revisiting this object's tag list later), "may be junk" (stuff that could be deleted), "irrelevant" (secondary images that are details of some others, to avoid cluttering search results), "ecclesia project" (what you're currently busy with), etc.

History slots are great for rarely used tags. First you manually pick some via the dialog, then they get assigned to these slots and you can recall them quickly, until they're "shifted away" by more recent selections.

Deck-specific slots are to be filled with thematic tags. For example, if you're a game designer and you are tagging animation references, then the 10 available slots may be filled with animation kinds, such as: idle, walk, run, attack, jump, looped, etc.

Once done with animations, you can switch to another deck used for describing pictured objects: cat, man, technology, furniture, nature, etc.

If your database is used for more than one kind of objects – for example, also video tutorials or phrase collection for writing stories, then you'll quickly grow out of 10 decks by 10 slots and start using a specific deck set for a specific kind of workflow.

Mass-Merging

Imagine you have done several imports with tags extracted from file names, using a regexp so that `ic_personal_video_black_36dp.png` (a typical Android resource file name) got tags `personal` and `video`. However, with one of the imports you put tags under a different parent (perhaps straight into the root), as a result getting two tags: `personal` in the root and `personal` under `Tags` – and now you need to remove the duplicates.

This is usually solved by using *Merge Into (This)* commands but if you have several thousands of such tags cleaning them up by hand would be ridiculous. Worry not – you can do that in one go if you work directly on the database (with care!) since merging is basically changing the `TagRowId` field's value of the `ObjectTags` table from the old (to-be-merged) tag to the new (combined/recipient) tag. Here's a PHP script doing that:

Database/Mass merge.php • Skip this example (#exQU) • Browse on GitHub • Adjustments (3)

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

$rootTagRowId = 1234;

$db = new PDO("sqlite:C:/Stagsi DB/Stagsi.sqlite");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$updateStmt = $db->prepare("UPDATE ObjectTags SET TagRowId = ? WHERE
TagRowId = ?");

$tags = $db->query("SELECT RowId, * FROM Tags WHERE ParentRowId =
$rootTagRowId")
  ->fetchAll(PDO::FETCH_OBJ);

$dupTags = [];

foreach ($tags as $tag) {
  $title = preg_replace('/^:+/u', '', $tag->Title);
  $ref = &$dupTags[mb_strtolower($title, 'utf-8')];
  $ref[] = $tag;
}

foreach ($dupTags as $tags) {
  if (count($tags) > 1) {
    $main = $tags[0];

    foreach ($tags as $tag) {
      if ($tag !== $main) {
        echo "merge $tag->RowId into $main->RowId", PHP_EOL;
        $updateStmt->bindValue(1, $main->RowId);
        $updateStmt->bindValue(2, $tag->RowId);
        $updateStmt->execute();
      }
    }
  }
}
```

To use it:

- First, move all potentially duplicate (to-be-merged) tags to under the same parent – but before

that prefix all such tags with a unique symbol like `:` to avoid `Title` conflicts:

```
UPDATE Tags SET TItle = ':' || Title WHERE ParentRowId = 1234
```

- Next, change `$rootTagRowId` in the script to the `RowId` of the parent tag containing all to-be-merged tags.
- Run the script. It will consider tags with zero or more `:` in the `Title` to be the "same" (e.g. `::SomeTag`) and merge them into one tag (this tag may happen to be with `:`s or without).
- Now tags are merged but old, duplicate tags remain as having no assigned objects. Open *Stagsi*, sort the tree by object count and delete them.
- Finally, remove `:` prefixes of the remaining tags (you may need to run the query multiple times if some tags have multiple `:`'s):

```
UPDATE Tags SET Title = SUBSTR(Title, 2) WHERE SUBSTR(Title, 1,
1) = ':' AND ParentRowId = 1234
```

Search

"Smart" Collections

Some programs (like *Adobe Bridge* and *Apple iTunes*) have two kinds of collections: "regular" and "smart". The first do not automatically refresh when keywords are updated while the second do.

Stagsi doesn't have "collections" per se. The closest analogy of "smart" ones is saved searches: whenever you repeat a search (saved or not – saving is just a faster way to enter the search expression) only actual image tags are checked, so searches are never out of date.

"Regular" collections in *Stagsi* are a special case of "smart" – create a tag like "My Collection 01" and assign it to the images of interest, then search for this tag alone.

Filter by Path

One common question of new *Stagsi* users is: how to find images that once were on some file system path?

Short answer: not possible.

Long answer: paths are remnants of the traditional file & folder hierarchy mindset. If you're trying *Stagsi*, it means you are not satisfied with that, and for a good reason. *Stagsi* doesn't work with paths.

Now, of course there is an "indirect" way. What are file paths and names? They're the kind of tags, if you think about it. Your birthday photos may be in `C:\Users\Eric\Junk\2018party` – and the last part, "2018party", is what identifies the collection. And you want to filter based on that – just as if it were a tag. So the real question is: can you convert path components into tags?

And yes, *that* is possible. The *Import* dialog offers tagging by wildcard and tagging by regular expressions (regexps). For example, `*\2018party\*` wildcard and `\\2018party\\` regexp can

both solve the birthday problem. Read those sections for more details.

Improving Relevance

Some users may find that highly relevant objects are lost within many not that relevant results. Some tips that may help:

- Create more tags and search for more specific tags. For example, if you have 1000 images of animals and you want only cats, create a "cat" tag under "animal" instead of putting everything into "animal", and search for "cat", not "animal".
- Utilize tag Weight. For example, picture of a smartphone in large is more relevant to "smartphone" query than a picture of a person with a smartphone, which is in turn more relevant than a photo of a hi-tech store show-casing a phone among other various devices. By assigning the first picture the weight of 3, the second 2, the third 1 you can make them appear in this order as long as your search query includes "smartphone" tag.
- Mark variations of the same image with some tag and exclude it in your searches. For example, burst photos can be all tagged with "minor" except one of them (or, in contrast, with "primary" tag for only one). Going further, you can create various tags for various groups of images, e.g. "Other party 2019-04 photos" and tag images with both "minor" and that tag. This way, locating variations is easier than with just one "minor" (or "primary") tag.
- Use tags and/or the `Random` column trick to put more relevant results before others.

Query Language

A good deal of the *Stagsi*'s versatility comes from its search query language. It's not complex, and by mastering it you will master *Stagsi* itself.

In the following explanation we are assuming this tag hierarchy:

```
+ people
  + male
  + elderly
  + female
+ nature
  + animals
    + cat
    + bird
  + landscape
  + winter
```

A search query consists of atoms – tags references. They can be in short and long forms. For the `cat` tag, the short form is just that – "cat", and the long form is: "nature/animals/cat", i.e. the combination of all parents' titles.

The short form can be used if the tag's own title is unique within the entire database. If we had another `cat` tag (under another parent tag) – we could not use this form, otherwise there would be no telling which of the `cat` tags we wanted to search for. But if another `cat` had no parents (was in the root), then writing just "cat" would refer to that root `cat` (since "cat" is both its short and long forms).

When you put two atoms – tags – next to each other (separating with a space), you are searching for *both* of them. To locate all images of cats on a winter landscape you do: "cat winter".

If you separate the tags with a pipe symbol rather than a space, you are searching for *either* of them. To locate images for all people, you can do: "male | female".

However, there's a better way. If you prefix a tag with a tilde, you look for that tag as well as *any of its child tags* at once: "~people". This is just a shorter form of writing "people | male | elderly | female". As you'd guess, to locate all animals you'd do "~animals" rather than enumerating every type of animal by hand. This works on any depth so that "~nature" includes all `animal`'s as well as `landscape` and `winter`.

Prefixing a tag with a dash "*blacklists*" it – objects with this tag are omitted from the results, even if they have some other tags that match the query. To look for non-winter cats you do: "cat -winter", and to look for all non-winter animals you do: "~animals -winter". You can even blacklist tags that are matched by the tilde: "people -elderly" gives you all people (`male` and `female`) who are not also `elderly`.

Prefixes can be *combined*. To look for all landscapes without any animals you do: "landscape -~animal".

But wait, there's more. You can *group tags* with brackets. If you want all landscapes without animals, unless they are a winter landscape – you do: "landscape (winter | -~animal)". This reads as: "take an image if it has `landscape` and either has `winter` or has no `animal` tag or one of its sub-tags" – i.e. "either (`landscape` and `winter`) or (`landscape` and no `animal` and no `animal` sub-tag)".

Finally, in the same situation you could use a tag, you can refer to a particular object using its `RowId` – just prefix it with `/`. This form is indispensible if you have pre-filtered objects (e.g. an external search tool using SQLite) and you only need to display them.

Other notes:

- You can't apply prefixes to brackets.
- You can give tag names in any character case – `ANIMAL`, `Animal`, `animal` are all referring to the same tag.
- If a tag name contains special symbols (like spaces and pipes) then you have to quote it: `"The Matrix"`. This equally applies to long forms: `"Top Movies"/"The Matrix"`.
- In your own program you can "extend" the standard syntax by creating terms of at least 2 titles, first of which is blank and second is non-numeric, for example: `/123/tagid` or `/tags/added-recently`.

Cheatsheet

```
short_form
long/form
"remember/to quote"
"long/form"/quoted
us all three
either | of | us
~me_and_children
-blacklist
-~not_me_or_children
me_plus (either | -us)
and (/123 | /456) "id"/s
```

Parsing Libraries

The repository with samples contains a very precise AST parser of *Stagsi*'s query syntax in multiple languages, including PHP, JavaScript, C# and plain C. Feel free to use it in your projects (public domain CC0 license).

If you plan on extending the query language with your own syntax, consider avoiding potential clashes in the future with the standard syntax by making sure your terms start with a blank title and have either 3 or more titles or 2 titles, of which first is non-numeric (doesn't consist entirely of 0-9 symbols).

For example, if you want to allow referring to tags by their `RowId`, it's safe to use `/tag/123` or `/123/tag` tag but not `/123` or `tag/123`.

Custom Sorting with Weights

Let us remember that tags may be also used for sorting, not just searching. The technique below allows practically unlimited "sorting profiles".

For example, you want images to be sorted by brightness or another aspect that is not exactly a "material tag" like "flower" or "sun" (even though brightness can be explained as "very bright", "a bit not too bright", etc. – using numeric qualifiers is much more intuitive). For this, create a tag – "brightness", and add it to all images but with a different Weight. This doesn't make images immediately sorted – you need to add "brightness" to your search query and make sure the relevance sorting is enabled (it is by default).

Custom Sorting with Random

Let's imagine you have a bunch of week day-themed objects and you want to see ones that correspond to the today's date first. It may sound bizarre (call it "flexible thinking"), but most script-friendly way to achieve this is with the `Random` column.

How does *Stagsi* produce random sorting? It's simple: every object in the database gets a random number when imported (and when *Randomize* or `/rerand` commands are used) which is used to compare it with others. Bigger `Random` – later sorting (when in ascending mode).

Nothing prevents us from hijacking this field and using for our purpose, temporary or not, for all database objects or only some. Another advantage is that we can "undo" our sorting by simple randomization.

Note: it's recommended to keep `Random` within this range: -2147483647 to 2147483647.

In this example, we're assuming you have a tag named "Weekdays" which has 7 child tags, each starting with a digit where 0 signifies Sunday (the British, rejoice) and 6 signifies Saturday. Let's throw in some *Powershell* and SQL (using *sqlite3.exe* from sqlite.org):

Misc/Sort by weekday.ps1 • Skip this example (#exCQ) • Browse on GitHub • Adjustments (1)

```
$db = "F:\Stagsi\DB\Stagsi.sqlite"
$day = (Get-Date).DayOfWeek.Value__
# First, reset order of all objects.
sqlite3 "$db" "UPDATE Objects SET Random = 0 WHERE RowId IN (
  SELECT o.RowId
    FROM Objects o
    JOIN ObjectTags ot
      ON ot.ObjectRowId = o.RowId
   WHERE ot.TagRowId IN (
```

```
      SELECT RowId
        FROM Tags
       WHERE ParentRowId = (SELECT RowId FROM Tags WHERE Title =
'Weekdays')
    )
)"
# Then, bump objects with the matching weekday.
sqlite3 "$db" "UPDATE Objects SET Random = -100 WHERE RowId IN (
  SELECT o.RowId
    FROM Objects o
    JOIN ObjectTags ot
      ON ot.ObjectRowId = o.RowId
   WHERE ot.TagRowId IN (
      SELECT RowId
        FROM Tags
       WHERE ParentRowId = (SELECT RowId FROM Tags WHERE Title =
'Weekdays')
         AND Title LIKE '$day%'
    )
)"
```

Custom Sorting with C# Plugin

Since *Stagsi* version 1802, `IPluginService` exposes a new method (see Plugins for general information):

```
void RegisterImageSorterV1(Func<List<IImageSortingV1>,
List<IImageSortingV1>> Sorter, ImageSorterV1)
```

Plugin-based sorting works in two phases: generation (once per each object in the database, potentially a lengthy process) and sorting (once per search, quick). Generator returns arbitrary data that is saved in the database under `Sorting\Plugin.Class.Name\Group\RowID` (`Group` = `RowId` divided by `FolderSize`). Sorting receives that data and may use it to reorder user's search results.

`IImageSorterV1` fields:

- `Title` – language-neutral string visible to the user to identify the sorting mode.
- `Generator` – an `Action<IImageSortingV1>` providing the data for subsequent sorting. Can be `null` if plugin needs no persistent data.
- `ParallelGeneration` – `true` by default, allowing *Stagsi* to call `Generator` simultaneously from multiple threads.

`IImageSortingV1` describes each search result and has these fields:

- `FilePath` – absolute path to the object's data file.
- `RowId` – the object's database ID.
- `Thumbnail` – a `BitmapSource` of the full-size thumbnail (from `t123.jpg`). Filled only if `Sorter` is being called and the object has an associated thumbnail.
- `Generated` – if `Generator` is being called: initally `null`, may be set to a `Stream` to be persisted; if `Sorter` is being called: the data previously returned by `Generator` (`null` if returned nothing or if `Generator` was `null`).

`Sorter` may modify the `List<>` it receives and may return the same object or a new one. If it deletes some members, they're re-added to the end (ordered by `RowId`). If it adds new members, they're ignored.

Randomize as a Marker

Even though the `Random` column may be signed, *Stagsi* itself only generates positive numbers when commands like *Randomize* and `/rerand` are used. This fact can be utilized to implement a "have seen" or another similar "yes/no" marker:

- At the beginning, reset the "marker" of all objects to "no" by executing `UPDATE Objects set Random = -1`.
- Open *Stagsi*, browse as usual, "marking" selected objects by calling *Randomize* from the context menu.
- Once done, examine "marked" objects by fetching their `RowId`'s (`SELECT RowId FROM Objects WHERE Random > 0`) and then searching for them in *Stagsi*:
  - Paste `RowId`'s returned by the `SELECT` query into a new *Notepad 2e* window
  - Use *Modify Lines* (`Alt+M`) to prefix them with `/`
  - Call *Join Lines* (`Ctrl+J`)
  - Copy the produced *Stagsi* query string with `Alt+C`
  - Paste to *Stagsi*'s query input to browse the "marked" objects

With this approach you cannot directly search for "marked" objects from within *Stagsi* but at least you don't have to close it to retrieve marked object `RowId`'s.

Thumbnails

Editing Thumbnails

Stagsi can generate thumbnails for only a handful of formats, mostly image-based (but you can write your own plugins). For others, none are generated and a placeholder image is used in the browser.

However, you can manually add thumbnails – and the dialog for doing so has numerous optimizations to make the task easy:

- The `F3` hotkey is your friend.
- If the dialog is invoked for multiple objects (in the selection), the resulting thumbnail is copied to all of them. Any object with *Custom Thumbnail* tag is used as a basis or, in lack of one, any at all with a thumbnail.
- If you have a folder of thumbnails, split-screen *Stagsi* and *Windows Explorer* and drag & drop files from the latter to the dialog.
- As an alternative to drag & drop, `Ctrl+C` (in *Explorer*) and `Ctrl+V` (in *Stagsi*'s dialog) can be used.
- Screenshots can be pasted the same way (`Ctrl+V`). For example, use `PrintScreen` or `Alt+PrintScreen` to "shot" a window (like a video player), `Ctrl+V` to paste and then crop the picture to better represent the content.

- Advanced image processing can be performed after initial pasting/cropping using the same `F3` hotkey (with the dialog opened). For example, *Fast Stone Image Viewer* is a powerful hotkey-friendly freeware tool that allows quickly modify (adjust levels, rotate, apply effects, etc.) and annotate graphics. Once saved in the external editor, the image is refreshed in the dialog (so keep it open or restart *Stagsi*).

Many Objects, Same Thumbnail

If you have imported 24 videos (of a single TV series) and *Stagsi* cannot generate thumbnails for this format, you end up with no thumbnails. But you want all these videos to have the same thumbnail. One option is to repeatedly set the same thumbnail via the *Edit Thumbnail* dialog. Another option is to paste user thumbnails directly into the data directory using a 3rd party batch renaming tool (*ReNamer* used here):

- Select all video objects in *Stagsi*.
- Copy their IDs using the *Object Properties* panel on the right.
- Close *Stagsi*.
- Prepare a thumbnail image file. Ensure its format matches your configured `ThumbFormat`.
- Copy the thumbnail file 24 times (simply holding `Ctrl+V` in *Windows Explorer*).
- Open the renaming tool, select those 24 thumbnail files and rename them according to the ID string copied earlier (in *ReNamer* the rule is called *User Input*) while also adding the `u` prefix to each line (which can be done manually or by adding a second rule called *Insert*).
- Finally, cut those thumbnails and paste into the correct numbered directory (or directories) within the data directory.

Power users may go further and utilize NTFS hardlinks to avoid wasting space for each copy as well as propagating changes done to one thumbnail to another. There is a convenient *Windows Explorer* shell extension schinagl.priv.at/nt/hardlinkshellext... that allows working with hard- and soft-links (symbolic links) like with regular shortcut files (`.lnk`), supporting drag & drop and other commands.

Another option suitable for scripting is the built-in *fsutil* command:

```
fsutil hardlink create C:\new.txt C:\existing.txt
```

The above command can be used in a script with a loop, or we can use *Notepad 2e*:

- Copy object IDs and close *Stagsi* as in the previous example.
- Paste IDs into a blank *Notepad 2e* window.
- Do *Modify Lines* (`Alt+M`) with the prefix text of `fsutil hardlink create C:\...\Database\1\u` and the append text of `.png C:\thumbnail.png` (adjusted to your needs). As a result, you will get a series of executable commands, one per each object:

  ```
  fsutil hardlink create C:\...\Database\1\u123.png
  C:\thumbnail.png
  fsutil hardlink create C:\...\Database\1\u124.png
  C:\thumbnail.png
  ...
  ```

- Save them as a `.bat` file somewhere and run by double clicking in *Windows Explorer* or using the *Launch* command (`Ctrl+L`).
- The `.bat` file and the original thumbnail (*C:\thumbnail.png*) are no more needed and can be deleted.

Batch Cropping

Stagsi stores crop state in the database as 4 floating-point numbers – relative distance from the top-left corner. For example, cropped top-right corner of the image has this state:

- `ThumbCropX` = 0.5
- `ThumbCropY` = 0.0
- `ThumbCropWidth` = 0.5
- `ThumbCropHeight` = 0.5

Let's suppose you have a large number of portraits of standing people. Because of too different aspect ratio, they might not be convenient to browse at small zoom (with small thumbnails). You can batch-crop all such thumbnails to only display the top third's horizontal center, i.e. X = 0.25, Y = 0, Width = 0.5, Height = 0.33.

We will set these crop settings using SQL (SQLite), but we also need to tell *Stagsi* that our objects need cropping – this is determined by the existence of `c` files. If all of your objects of interested already have cropped thumbnails, then some steps can be skipped, otherwise their stub files will need to be created.

- Copy object IDs and close *Stagsi* as in the previous example.
- Paste IDs into a blank *Notepad 2e* window and then into the second blank window.
- Switch to one of the windows. Do *Modify Lines* (`Alt+M`) with the prefix text of `,` and do *Join Lines* (`Ctrl+J`). Delete the unnecessary comma in the beginning.
- You now have a comma-separated list of `RowId`'s.
- Compose the SQL query of this form, putting the ID list inside the brackets (line breaks can be added anywhere for convenience):

```
UPDATE Objects
   SET ThumbCropX = 0.25, ThumbCropY = 0.0,
       ThumbCropWidth = 0.5, ThumbCropHeight = 0.33
 WHERE RowId IN (...)
```

- Save the SQL query to some file and copy its path (e.g. using *Notepad 2e*'s *Run* window – `Ctrl+R`, then `Ctrl+C` to copy).
- Run that SQL on the database. Here's how to do it with the official console *sqlite3* tool (but any other can be used):

```
sqlite3.exe "C:\...\Database\Stagsi.sqlite" <query.txt
```

- Create crop stub files:
  - Switch to the second *Notepad 2e* window with the list of IDs.
  - Use *Modify Lines* (`Alt+M`) with the prefix text of `echo >C:\...\Database\1\c` and the append text of `.png`. Replace the corresponding parts accordingly. As a result, you will get a series of executable commands, one per each object:

    ```
    echo >C:\...\Database\1\c123.png
    echo >C:\...\Database\1\c124.png
    ...
    ```

  - Save it as a `.bat` file somewhere and run by double clicking in *Windows Explorer* or using the *Launch* command (`Ctrl+L`).
- The query file and the `.bat` file are no more needed and can be deleted.
- Run `/rethumb` to regenerate the cropped versions according to our changes.

Permanent Cropping

If you're making a database of computer programs (or games) then you might be assigning screenshots (made with `PrintScreen`) to each object (a program or a game) and cropping them using *Edit Thumbnail* dialog. This works but full-size screenshots remain on disk as `u`-files (see ThumbU).

If you don't plan on ever changing the crop region, you can free up the space by making the cropped version permanent (i.e. "full-size") by:

- Renaming `c` thumb files into `u` (if `u` exists then overwriting it).
- Setting these fields in the `Objects` table to `0.0`:

  ```
  ThumbCropX
  ThumbCropY
  ThumbCropWidth
  ThumbCropHeight
  ```

Here is a helper PHP script:

Thumbnails/Permanent crop.php • Skip this example (#exCA) • Browse on GitHub • Adjustments (3)

```
<?php
$rowidsFile = 'c:/rowids.txt';
$dataPath = 'c:/stagsi/db';

$ids = [];

foreach (file($rowidsFile) as $id) {
  if ($id = trim($id)) {
    $path = $dataPath.'/'.floor($id / 1000).'/';
    rename("$path/c$id.png", "$path/u$id.png");
    $ids[] = $id;
  }
}
```

```
echo 'UPDATE Objects SET
ThumbCropX = 0.0,
ThumbCropY = 0.0,
ThumbCropWidth = 0.0,
ThumbCropHeight = 0.0
WHERE RowId IN (', join(',', $ids), ')';
```

It takes `RowId`'s of affected (cropped) objects from a text file (`$rowidsFile`), performs step 1 above (renames `c` into `u`) and outputs a SQL statement that you should run against that database using *SQLite Browser* or command-line `sqlite3.exe` (this is step 2).

Stagsi must not be running while carrying this out.

Batch Transformation

Let's imagine we want to make some of our thumbnails grayscale. This can be done with *ImageMagick* and *cmd.exe* kun-fu.

Here's a batch file that converts all standard thumbnails (of formats that *Stagsi* recognizes):

{} Thumbnails/Grayscale using ImageMagick.bat • Skip this example (#exNX) • Browse on GitHub • Adjustmen

```
setlocal ENABLEDELAYEDEXPANSION
cd /D C:\...\Database
for /R %%i in (t*.*) do (
set F=%%i
convert "%%i" -set colorspace Gray -separate -average "u!F:~1!"
)
```

Just close *Stagsi* and run this batch file (replacing the database path accordingly).

Text Overlay with Ruby

Often a good idea is to annotate a thumbnail with some short caption. The below Ruby script (using *RMagick*, a Ruby interface to *ImageMagick*; rmagick.github.io) processes a file in a simple which could be produced by a spreadsheet program like *MS Excel* or by hand, where each line is a text to be drawn, starting with a numerical object's `RowId` followed by a space, comma or semicolon. For example:

```
13, Cool GFS
15  FF XII video
87; Sketching guide
```

Here's the script that etches the string from the string file on a thumbnail of every object in the database (note how it correctly copies the `t`-file if neither `u` nor `c` are found – `t`-files should never be edited):

{} Thumbnails/Text overlay.rb • Skip this example (#exHO) • Browse on GitHub • Adjustments (2)

```
require 'rmagick'
require 'fileutils'

data_path = 'C:/Stagsi/Database'
```

```
IO.read('strings.csv')
    .scan(/^\s*(\d+)(\s*[,;]\s*|\s+)(\S.*)$/)
    .each { |a|
        rowid, delim, str = a
        mask = "#{data_path}/#{(rowid.to_i/1000).floor()}/[cut]#
{rowid}.*"

        files = Dir.glob(mask)
            .map { |f| [File.basename(f)[0], f] }
            .to_h

        if files['c'] then
            file = files['c']
        elsif files['u'] then
            file = files['u']
        elsif files['t']
            dir, file = File.split(files['t'])
            file = "#{dir}/" + file.sub(/./, 'u')
            FileUtils.copy files['t'], file, :verbose => true
        else
            next
        end

        im = Magick::Image.read(file).first
        text = Magick::Draw.new

        # Actual overlaying happens here.
        text.annotate(im, 0, 0, 0, 0, str) {
            self.gravity = Magick::SouthGravity
            self.pointsize = 50
            self.fill = 'black'
            self.font_weight = Magick::BoldWeight
            self.stroke = 'white'
            self.stroke_width = 3
        }

        im.write(file)
        im.destroy!
    }
```

Supported Formats and Plugins

Stagsi can deal with any kind of files, even those it doesn't support. "Support" in this case means automatic thumbnail generation and meta-data extraction. For "unsupported" formats you'll see placeholder images in the browser, which you can easily replace by hand (see that section).

The `ImportUnknownFormats` setting (enabled by default) controls if *Stagsi* skips unsupported formats during import or not.

The actual support is provided by plugins (see Plugins for writing your own). Standard plugins coming with *Stagsi* by default include:

- `WpfPlugin.dll` – supports popular graphic formats using native C#/WPF capabilities with best performance: BMP, GIF, animated GIF, PNG, JPEG.
- `PsdPlugin.dll` – supports most PSDs but may fail on those using advanced Photoshop features – in this case download the *Magick* plugin.
- `XamlTunePlugin.dll` – supports most SVGs. Download the *Magick* plugin for complete support.

More standard plugins that you can download from go.soletude.ca/stagsi/forum:

- `MagickPlugin.dll` – a graphics library (*ImageMagick*) with solid support for a wide range of formats: PSD, SVG, CUR, EMF, ICO, PCX, TGA, TIFF, TTF, WMF.

These meta-data formats are supported by means of *MetadataExtractor*: EXIF, IPTC, XMP.

Custom Thumbnails – Manual Generation

To "support" a format, you could write a tool to create thumbnails for data files that are missing them. Unlike writing a plugin, you can use any language (not just C#) but you will have to run this program or script by hand (it won't respond to `/rethumb`).

Here's a C/C++ program that renders file type icon for every object that has no thumbnail (`t`-automatic or `u`-user-assigned) – assuming `ThumbFormat` is PNG:

{} Thumbnails/Generate from file icon.c • Skip this example (#exEO) • Browse on GitHub

```
// Compile with: cl /TP this.cpp_or_.c /link gdiplus.lib shell32.lib

#include <windows.h>
#include <strsafe.h>
#include <Gdiplus.h>

bool has_thumb(wchar_t *data_path, wchar_t *dir, wchar_t *prefix,
long rowid) {
    wchar_t mask[MAX_PATH];
    StringCchPrintfW(mask, MAX_PATH, L"%s\\%s\\%s%ld.*", data_path,
dir, prefix, rowid);
    WIN32_FIND_DATAW find = {0};
    HANDLE handle = FindFirstFileW(mask, &find);
    if (handle == INVALID_HANDLE_VALUE) {
        return false;
    }
    FindClose(handle);
    return true;
}

// Procedural-style GDI+. Close to black magic.
// https://forum.antichat.ru/threads/268293/
bool hicon_to_png(wchar_t *dest, HICON icon) {
    // http://www.masm32.com/board/index.php?topic=5782.0
    // Type: image/bmp,   GUID: {557CF400-1A04-11D3-9A73-
0000F81EF32E}
    // Type: image/jpeg,  GUID: {557CF401-1A04-11D3-9A73-
0000F81EF32E}
    // Type: image/gif,   GUID: {557CF402-1A04-11D3-9A73-
0000F81EF32E}
    // Type: image/tiff,  GUID: {557CF405-1A04-11D3-9A73-
0000F81EF32E}
    // Type: image/png,   GUID: {557CF406-1A04-11D3-9A73-
0000F81EF32E}
    GUID png_format = {0x557CF406, 0x1A04, 0x11D3, 0x9A, 0x73, 0x00,
0x00, 0xF8, 0x1E, 0xF3, 0x2E};
    Gdiplus::GdiplusStartupInput si = {0};
    si.GdiplusVersion = 1;
    ULONG_PTR token;
    int res = Gdiplus::GdiplusStartup(&token, &si, NULL);
    if (!res) {
        Gdiplus::GpBitmap *bmp;
        res = Gdiplus::DllExports::GdipCreateBitmapFromHICON(icon,
```

```
&bmp);
        if (!res) {
            res = Gdiplus::DllExports::GdipSaveImageToFile(bmp,
dest, &png_format, NULL);
            Gdiplus::DllExports::GdipDisposeImage(bmp);
        }
        Gdiplus::GdiplusShutdown(token);
    }
    return !res;
}

int wmain(int argc, wchar_t** argv) {
    wchar_t *data_path = argv[1];

    wchar_t dir_mask[MAX_PATH];
    HRESULT o = StringCchPrintfW(dir_mask, MAX_PATH, L"%s\\*",
data_path);
    WIN32_FIND_DATAW dir = {0};
    HANDLE dir_handle = FindFirstFileW(dir_mask, &dir);

    if (dir_handle == INVALID_HANDLE_VALUE) {
        return 1;
    }

    do {
        wchar_t *end = dir.cFileName;
        while (*end && *end >= L'0' && *end <= L'9') { end++; }
        if (*end) { continue; }

        wchar_t file_mask[MAX_PATH];
        StringCchPrintfW(file_mask, MAX_PATH, L"%s\\%s\\*",
data_path, dir.cFileName);
        WIN32_FIND_DATAW file = {0};
        HANDLE file_handle = FindFirstFileW(file_mask, &file);

        if (file_handle == INVALID_HANDLE_VALUE) {
            continue;
        }

        do {
            // strtol()/wcstol() allow leading whitespace and sign.
            if (*file.cFileName < L'0' || *file.cFileName > L'9') {
                continue;
            }

            wchar_t *end;
            long rowid = wcstol(file.cFileName, &end, 10);
            wchar_t data_file[MAX_PATH];
            StringCchPrintfW(data_file, MAX_PATH, L"%s\\%s\\%s",
data_path, dir.cFileName, file.cFileName);
            if (*end == L'.') {
                // Not a link file.
            } else if (!*end) {
                HANDLE link_handle = CreateFileW(data_file,
GENERIC_READ, 0, NULL, OPEN_EXISTING, 0, NULL);
                DWORD link_read;
                char buf[MAX_PATH];
                // MultiByteToWideChar() won't write '\0'.
                memset(data_file, 0, sizeof(data_file));
                if (link_handle == INVALID_HANDLE_VALUE ||
                    !ReadFile(link_handle, buf, sizeof(buf) - 1,
&link_read, NULL) ||
                    !link_read ||
                    !MultiByteToWideChar(CP_UTF8, 0, buf, link_read,
data_file, MAX_PATH)) {
```

```
                continue;
            }
            CloseHandle(link_handle);
        } else {
            continue;
        }

        if (has_thumb(data_path, dir.cFileName, L"t", rowid) ||
            has_thumb(data_path, dir.cFileName, L"u", rowid)) {
            continue;
        }

        SHFILEINFOW info = {0};
        SHGetFileInfoW(data_file, 0, &info, sizeof(info),
SHGFI_ICON | SHGFI_LARGEICON);

        wchar_t thumb_file[MAX_PATH];
        StringCchPrintfW(thumb_file, MAX_PATH,
L"%s\\%s\\u%ld.png", data_path, dir.cFileName, rowid);

        hicon_to_png(thumb_file, info.hIcon);
    } while (FindNextFileW(file_handle, &file));

    FindClose(file_handle);
} while (FindNextFileW(dir_handle, &dir));

FindClose(dir_handle);
}
```

Custom Thumbnails – Via Scripts

If you have a small number of objects that need custom thumbnails, a more efficient approach is to apply them from within *Stagsi* on demand.

First, write a script that extracts object ID from the path it's given (the file's name is `u123.ext` where `123` is the ID – see DataNumbered), retrieves required info by that ID (e.g. the object's main file), generates a thumbnail image and saves it to that path (`argv[1]`).

Second, invoke *Stagsi*'s thumbnail dialog with `F3`, then press `F3` again to call an external editor. A list of programs will appear (unless you have configured it to always call a specific program) – add your script to this list. Now you can quickly call it for a particular object that needs custom thumbnails, working like specialized Tools.

Custom Thumbnails – C# Plugin

Plugins are loaded (in arbitrary order) from two folders: `Plugins` near *Stagsi*'s EXE and `Plugins` in the database. A plugin may be either a `.dll` or a `.cs` file. `.cs` is a C# source code file compiled and loaded on runtime as if it was a pre-compiled DLL, with implicit references to DLLs already loaded into *Stagsi* and to all DLLs in all `Plugins` folders, and with explicit reference from `// ref: c:\path\to.dll` comment(s) in the end of the file (ignoring blank lines).

Note: our community repository go.soletude.ca/stagsi/cookbook has a few plugins which show how to wrap a C++ library into a C# plugin, e.g. `psd2png`.

Each plugin must define one or more classes containing a `static` function named `StagsiPlugin()` returning `void` and accepting a single argument typed

`Soletude.Stagsi.Plugins.IPluginService` exposing this method (version 1802 added SortingPlugins):

```
void RegisterImageLoaderV1(Func<Stream, IImageLoadingV1,
BitmapSource>, ImageLoaderV1)
```

`ImageLoaderV1` specifies additional configuration and currently contains just one property: `FileExtensions` (an array of `string`) – typical file name extensions (without the period) for the kind of format this handler is able to process. This is used in file dialogs and internal optimizations and may be empty. First string is considered the "canonical" extension visible as a group title for other extensions (aliases) – e.g. `JPG`, alias: `JPEG`.

When *Stagsi* needs to decode a file into an image data it calls all registered handlers (in arbitrary order) giving them access to the file's raw data (`Stream`) and its information (`IImageLoadingV1`). If a handler is unable to process the data it returns `null`.

`Stream` has its pointer at the beginning of the data and may or may not correspond to a local file. The handler may close it (`CanRead()` returning `false` – may happen if the stream's handle is transferred to native code) or advance regardless if `null` is returned or not.

Note: regardless of `FileExtensions` a handler may be called for any input file (which may not be a local file or may have an invalid extension, e.g. `.png` instead of `.gif`) and so must check if the input `Stream` contains data that it recognizes, e.g. using "magic signatures" (see the popular `file` utility astron.com or this common signature repository www.garykessler.net/Library/file_sigs.h...).

`IImageLoadingV1` fields:

- `FilePath` (`string`) – absolute path to the original file; may be `null`. Avoid using it as a final data format check because file extensions may be wrong.
- `QueryOnly` (`set`) – requested information; if `null` then complete decoding must be performed, else only these fields may be set and a 0x0 `BitmapSource` returned to indicate a supported format. It's an optimization – the handler may ignore it loading the image anyway or it may set more fields than requested.
- `Width` and `Height` (`double`) – desired maximum dimensions of the returned `BitmapSource`. *Stagsi* scales the result as appropriate in any case so this only makes sence for special scaling algorithms, e.g. if the original graphics is vector. If 0, the original dimensions must be used (if there are none then the plugin is free to choose them). The handler must set both to the original image's dimensions on return.

  Note: normally you would want to preserve the aspect ratio of the original image instead of distorting it to fit the given `Width`/`Height`.

- `IsAnimated` (`bool`, ignored if `null` is returned) – may be set by the handler if `Stream` "contains" more than one frame (an animated GIF, video, etc.) to assign the `Animated` system tag.
- `Format` (`string`, ignored if `null` is returned) – may be set to specify the "format" system tag, typically the "canonical" file extension for this kind of data (e.g. `TIFF`, not `TIF`). If blank then the original file's extension is used, or `dat` if it's unavailable.

Conventions for plugin writers (assuming a hypothetical plugin named "WebPage"):

- Add "Plugin" to the DLL's file name: `WebPagePlugin.dll`. Many plugins require an external library (e.g. `Magick.dll` for an ImageMagick plugin) and this makes it clearer which one is a *Stagsi* plugin and which is its dependency.
- Return the image as is if your plugin doesn't handle resizing specially (e.g. using xBRZ or resizing an originally vector image) to let *Stagsi* resize it as appropriate according to program settings (e.g. sharply for small images).
- Return a raw bitmap, not an encoded image (e.g. `PngBitmapEncoder`) to avoid the overhead of first decoding it in the *Stagsi* core and then encoding to the user's `ThumbFormat` (e.g. JPEG).

You can debug your plugin with Visual Studio's *Attach to process* command, setting a breakpoint in your plugin's code and invoking an operation in *Stagsi* that calls it (e.g. *Copy* in *Object Browser*).

With all this, a sample plugin rendering text files into the thumbnail's box (save as `Plugins\TxtPlugin.cs` and restart *Stagsi*):

Thumbnails/TxtPlugin.cs • Skip this example (#exGP) • Browse on GitHub • Adjustments (6)

```
using System.Globalization;
using System.IO;
using System.Text;
using System.Windows.Media.Imaging;
using System.Windows.Media;
using System.Windows;

namespace Soletude.Stagsi.Plugins
{
    public class Txt
    {
        private const string TxtFormatName = "txt";
        private const string EndLineChars = "\r\n";
        private const string RenderFontName = "Arial";
        private const int MaxLength = 10000;
        private const double DefaultSize = 200;
        private static readonly string[] InvalidStrings = {"\0"};

        public static void StagsiPlugin(IPluginService service)
        {
            service.RegisterImageLoaderV1(RenderTxt, new
ImageLoaderV1 { FileExtensions = new[] { TxtFormatName }});
        }

        private static BitmapSource RenderTxt(Stream input,
IImageLoadingV1 parameters)
        {
            var len = input.Length;
            if (input.Length > MaxLength)
            {
                len = MaxLength;
            }
            byte[] buf = new byte[len];
            input.Read(buf, 0, buf.Length);
            string allText = Encoding.UTF8.GetString(buf);

            foreach (var v in InvalidStrings)
            {
                if (allText.Contains(v))
                {
```

```
                    // Binary data - skip.
                    return null;
                }
            }

            allText = allText.Replace(EndLineChars, " ");
            var txt = new FormattedText(allText,
CultureInfo.CurrentCulture, FlowDirection.LeftToRight, new
Typeface(RenderFontName), 10.0, Brushes.Black);
            return WriteTextToImage(txt, new Point(0, 0),
parameters);
        }

        public static BitmapSource WriteTextToImage(FormattedText
text, Point position, IImageLoadingV1 parameters)
        {
            var wid = parameters.Width;
            if (wid == 0 || double.IsNaN(wid))
            {
                wid = DefaultSize;
            }

            var hei = parameters.Height;
            if (hei == 0 || double.IsNaN(hei))
            {
                hei = DefaultSize;
            }

            var visual = new DrawingVisual();
            var prop = hei / wid;
            hei = (int) (wid * prop);

            text.MaxTextWidth = wid;
            using (var dc = visual.RenderOpen())
            {
                dc.DrawRectangle(Brushes.White, null, new Rect(new
Size(wid, hei)));
                dc.DrawText(text, position);
            }

            var target = new RenderTargetBitmap((int) wid, (int)
hei, 96.0, 96.0, PixelFormats.Default);
            target.Render(visual);

            parameters.Format = TxtFormatName;

            return target;
        }
    }
}
```

Browser Copy

This is one of those seemingly unassuming commands that you find more and more uses for every day.

When you perform a copy on a selection in the browser, the clipboard receives data of 3 types:

- Plain text string – for pasting to another *Stagsi* process and/or performing advanced tasks on.
- Image data – of the first object recognized as an image (for which *Stagsi* can generate thumbnails using plugins). Can be pasted into any graphic editor like *MS Paint*, *Photoshop*, etc. Can be also

done with simple drag & drop allowing to use *Stagsi* as a graphics library – try dropping into *MS Word*.

- *Windows Explorer* file list – for pasting into some folder on your system. Much like the *Export* command. Drag & drop can be used here as well.

The first type is the main source of this command's versatility – we'll even call it "the *Stagsi* interchange format". The string it creates looks like this:

```
Stagsi JSON 1 {JSON}
```

First three parts are constants ("1" is the format version number), acting as a marker: if a string in the clipboard begins with this sequence then it's assumed to be "paste-able".

The fourth part is a multi-line JSON object which mimics the database structure with a few extra fields. This object holds information about the selection and is independent of the *Stagsi* process that has created it (i.e. this string can be pasted to another *Stagsi* process after the originating *Stagsi* was closed).

- `Objects.FilePath` – path to the object's file (which was imported); for linked mode, this is the path to the "real" file (outside of the database).
- `Objects.FileLink` – path to the link data file (extensionless) if the object was imported in linked mode, otherwise `null`.
- `Objects.CustomThumb` – path to a user-assigned custom thumbnail or `null` if there's none.
- `Tags.IconPath` – path to the tag's icon or `null` if there's none.

Synthesized Paste

Often you need to import data from another tool, e.g. from an XML dump of a custom format. One way is to write to `Stagsi.sqlite` directly as we have already demonstrated:

- Works even if you have 100,000s of entries to import.
- But your language must support SQLite (or calling `sqlite3.exe`).
- Usually issuing SQL queries is not very convenient.
- *Stagsi* must be closed while the database is being updated.
- Your script must be updated when database schema changes in a future *Stagsi* release.

Another way is using the JSON format produced by the *Copy* command where you produce this JSON and *Stagsi* consumes it by means of *Paste*:

- Uses system clipboard and so will fail on extremely large number of entries (that can't fit into the clipboard).
- Because all entries have to be packed into the same JSON object, uses much more memory than direct SQLite interop.
- No dependency on an SQLite library or its executable, or SQL in general.
- You do need *Stagsi* to actually incorporate your changes but it doesn't need to be restarted to see them.

- The format is more relaxed and future-proof. Most fields can be omitted to use the defaults (e.g. `Tags.Color`, `ObjectTags.Weight`, `Objects.Random`).

Below is a script converting XML data from MyAnimeList.Net into *Stagsi*'s JSON format creating a temporary `.url` file for each object linking to that series' page on the website. For *Paste* to work, the JSON must be prepended by the version info (see above, e.g. `Stagsi JSON 1`). You can use `clip` to copy a string to the clipboard from the console.

User's score (series rating) appears both as a tag (under `Score`) and a Weight so that to search results by this score you just include the `~Score` into your search query (and enable *Relevant first*).

Import/Convert MAL.php • Skip this example (#exLQ) • Browse on GitHub

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

mb_internal_encoding('UTF-8');

list($self, $file) = $argv + ['', ''];

if (!is_file($file)) {
  echo "Usage: php $self MyAnimeList.xml";
  exit(1);
}

$xml = new DOMDocument;
$xml->load($file);
$xpath = new DOMXPath($xml);
$objects = $info = [];

$childToTag = [
  'series_type'     => 'Type',
  'series_episodes' => 'Episodes',
  'my_score'        => 'Score',
  'my_status'       => 'Status',
];

foreach ($xpath->evaluate('//anime') as $anime) {
  $object = [];
  foreach ($anime->childNodes as $child) {
    $ref = &$childToTag[$child->nodeName];
    if ($ref) {
      $object[$ref] = $child->textContent;
    } elseif ($child->nodeName === 'series_title') {
      $info[count($objects)]['title'] = $child->textContent;
    } elseif ($child->nodeName === 'series_animedb_id') {
      $info[count($objects)]['id'] = $child->textContent;
    }
  }
  $objects[] = $object;
}

$json = ['Objects' => [], 'Tags' => [], 'ObjectTags' => []];
$temps = [];

foreach ($objects as $id => $object) {
  $temps[] = $temp = tempnam(sys_get_temp_dir(), '').'.url';
  file_put_contents($temp, "
[InternetShortcut]\nURL=https://myanimelist.net/anime/".$info[$id]
['id']);
```

```
  $json['Objects'][] = [
    'CreationTime' => time(),
    'FilePath' => $temp,
    'Random' => mt_rand(),
    'RowId' => $id + 1,
    'Title' => $info[$id]['title'],
  ];

  foreach ($object as $tag => $value) {
    $tag = findCreateTag($json['Tags'], [$tag, $value]);
    $json['ObjectTags'][] = [
      'ObjectRowId' => $id + 1,
      'TagRowId' => $tag['RowId'],
      'Weight' => $tag === 'Score' ? $value : 0,
    ];
  }
}

echo 'Stagsi JSON 1 '.json_encode($json, /*JSON_PRETTY_PRINT+*/ 0);

function findTag(array $tags, $title, $parent = null) {
  foreach ($tags as $ttag) {
    if ($ttag['ParentRowId'] === $parent and
        mb_strtolower($ttag['Title']) === mb_strtolower($title)) {
      return $ttag;
    }
  }
}

function findCreateTag(array &$tags, array $path) {
  static $nextID = 1;
  $parent = null;

  foreach ($path as $ptag) {
    $ttag = findTag($tags, $ptag, $parent);
    if (!$ttag) {
      $tags[] = $ttag = [
        'CreationTime' => time(),
        'ParentRowId' => $parent,
        'RowId' => $nextID++,
        'Title' => (string) $ptag,
      ];
    }
    $parent = $ttag['RowId'];
  }

  return $ttag;
}
```

Merge and Split Databases

Two obvious operations for *Copy/Paste* between *Stagsi* processes are to merge and split databases.

Let's imagine you have a big database and you want to extract a portion of it, for example, for sharing with someone else:

- Select files that you want to "extract". If you can't get them all into the same list (e.g. because they need to be located using different searches), then add a temporary tag to such files, perform a search for this tag, select (`Ctrl+A`) and copy (`Ctrl+C`) all results, then delete the tag.

- Create a new database and open it. This can be as simple as creating an empty directory and dragging & dropping it from *Windows Explorer* into a running *Stagsi*'s window or onto its icon on the desktop.
- Paste (`Ctrl+V`) results copied earlier into this new database's window.

Now let's imaging you have two databases, perhaps created at different times, that you want to merge into one (it's generally easier to manage one database than multiple smaller ones):

- Open the first database you're going to merge.
- If you want to later tell from which database an object has came from, add some tag to all objects in this database.
- Open an empty search tab (blank search query) and select (`Ctrl+A`) and copy (`Ctrl+C`) all objects.
- Switch to the main ("big") database. It can be a brand new database or one of the to-be-merged (in this case all other databases are put into it). Do paste (`Ctrl+V`).
- Close the first database, open the second and repeat the same steps until you merge all of them.

Confirming Changes (Diff)

Let's imagine you are performing a large clean-up of your database, adding and removing tags from hundreds of objects. In the end of the day, you want to confirm what exactly was changed before you commit the version.

The *Copy* command can be used to quickly convert part of a database into a textual representation. And for text, there are tools called "diff"/"merge" viewers: they take 2 or more files, compare them line-by-line and display the differences. For Windows, *WinMerge* can be used, or a similar tool from virtually any version-control system (GUIs for *git*, *SVN*, etc.). Linux has a built-in console command *diff*, or a GUI called *meld*.

- Before beginning to modify the database, run a search on the objects that will be affected (or search for empty string if all of them will be).
- Select all (`Ctrl+A`) and Copy (`Ctrl+C`).
- Open a text editor such as *Notepad* and Paste (`Ctrl+V`), then save (`Ctrl+S`) to a text file, say, `before.txt`.
- Perform the modifications.
- Repeat the steps in the beginning: run the search, Select all results, Copy and Paste to a new text document, and then save as another file, say, `after.txt` (alongside the first one).
- Select both files in *Windows Explorer*.
- Bring up the files' context menu and select the diff command of your choice.
- Go through all the changes to confirm them, possibly switching back to *Stagsi* to amend things.

Multiple Clipboards

The text inserted into the clipboard by the *Copy* command is self-contained – the original *Stagsi* process is not required for Paste into another *Stagsi* process to work.

This means you can paste results of the *Copy* command into a text editor like *Notepad* leaving its window open, perform another *Copy* followed by *Paste* into a new window, etc. and then go back to any of those windows, *Copy* everything and *Paste* into *Stagsi* later.

In an extreme case, you can even save the copied data to a text file and reuse it much later, even after a reboot or weeks of regular usage. It will work as long as file paths (to data files, thumbnail files, etc.) remain valid.

AutoHotKey

AutoHotKey, or *AHK* for short, is a powerful automation utility based on scripts. Here's a script that displays a tooltip with the number of copied objects imported in linked mode:

Misc/Copy tooltip.ahk • Skip this example (#exNF) • Browse on GitHub

```
#Persistent
OnClipboardChange:
StringLeft, Start, Clipboard, 7
if (Start = "Stagsi ")
{
  StrReplace(Clipboard, """FileLink"": null,", , CopiedCount)
  StrReplace(Clipboard, """FileLink"": ", , TotalCount)
  LinkedCount := TotalCount - CopiedCount
  ToolTip %LinkedCount% out of %TotalCount% objects are linked
  Sleep 5000
  ToolTip
}
return
```

Node.js

Even though JSON is native to most scripting languages, many of them – like PHP and *Node.js* (JavaScript) – don't make it easy to access the system clipboard (mostly because there's no cross-platform way of doing so).

The simplest way is to use an external tool. In the following script we're using *nircmd* (which should be in your %PATH%) to obtain *Stagsi* clipboard text and output the total size of all copied object files:

Misc/Copied file size.js • Skip this example (#exRA) • Browse on GitHub

```
var file = require('fs').mkdtempSync(os.tmpdir() + '/') +
'/clip.txt'
require('child_process').execSync('nircmdc64 clipboard writefile "'
+ file + '"')
var clipboard = require('fs').readFileSync(file, {encoding: 'utf8'})
var [, data] = clipboard.match(/^Stagsi JSON \d+ (\{.+)$/s)
var sum = JSON.parse(data).Objects.reduce((cur, obj) => cur +
obj.FileSize, 0)
console.log(`Total size of all objects: ${sum} bytes`)
```

Notepad 2e

Notepad 2e is a bare-bone text editor with advanced data manipulation capabilities aimed at quick execution of common one-time operations (in contrast to tools like *sed* and *vim* which shine on

repetitive tasks).

For example, if you want to prepend "Game" to titles of all objects before they're pasted:

- *Paste* (`Ctrl+V`) *Stagsi* data into a blank *Notepad 2e* window.
- Do *Replace* (`Ctrl+H`) on the entire document, from string `"Title": "` to string `"Title": "Game:`.
- Do *Copy All* (`Alt+C`) to clipboard (which is a single-keystroke version of *Select All* followed by *Copy*) and paste to the second *Stagsi*'s window.

Another example, to obtain a list of objects that came (`OriginalPath`) from the `E:\Important` directory or its subdirectories:

- *Paste* (`Ctrl+V`) *Stagsi* data into a blank *Notepad 2e* window.
- Do *Replace* (`Ctrl+H`) on the entire document, from string `},` to string `},\n` with enabled *Transform backslashes* option. This adds a blank line after each individual object.
- Do *Select all* (`Ctrl+A`) followed by *Join Paragraphs* (`Ctrl+Shift+J`). This makes it so that each object takes 1 line, not more.
- Open *Find* (`Ctrl+F`), enter the search string of *E:\Important* and press *Grep* (`Alt+G`). Now you have 1 line per each object which has this string somewhere in its properties (you can easily adjust the search string to be more precise, such as to "*OriginalPath*": "*E:\Important*).
- Next actions depend on what you want to do with this data. Usually you'd want a list of IDs for searching for these objects in *Stagsi*. For this, do *Replace* from string `.*"rowid": |,.*` to blank string with enabled *Regular expression search*. The expression stands for "replace everything before 'rowid' and after comma with nothing, i.e. delete".
- As a result, you got 1 ID per line. To search for them in *Stagsi*, they need to be converted to 1 line of form `/id1 | /id2 | /id3 | ...`. Do *Modify Lines* (`Alt+M`) with the prefix text of `| /` and then *Join Lines* (`Ctrl+J`). As a result, you'll get an extra pipe at the beginning which you just delete by hand.
- Finally, do *Copy All* (`Alt+C`) to clipboard and paste to *Stagsi*'s search input.

Spreadsheet

Suppose you want to make a neat table of your objects, perhaps for sorting, filtering or some other manipulations that spreadsheet programs are great at.

In this example we're going to combine *Notepad 2e* magic with a lightweight alternative to *MS Excel* (1 EXE file, less than 2 MiB) called *Spread32*.

First, we need to format *Stagsi* data so that it becomes 1 line per object, with object properties separated with a *Tab* character:

- *Paste* (`Ctrl+V`) *Stagsi* data into a blank *Notepad 2e* window.
- Remove all arrays except `Objects` by seeking the start of this array using *Find Matching Brace* (`Ctrl+B`), then deleting the part before it and the part after it (again with `Ctrl+B`).
- An example of text you should have at this point:

```
Stagsi JSON 1 {
   "ObjectTags": [
      ...
   ],
   "Objects": [

      {
        "CreationTime": "1554727353172",
        "CustomThumb": false,
        "CustomThumbPath": null,
        "FileLink": null,
      ...

   ],
   "Tags": [
      ...
   ]
}
```

- Do *Replace* (`Ctrl+H`), from string `[^:]+:` to string `\t` with enabled *Regular expression search*. This replaces property names with *Tab* characters, which will be later recognized by the spreadsheet program.
- Do another *Replace* (`Ctrl+H`), from string `\{$` to blank string with enabled *Regular expression search*. This creates blank line between each object.
- Do *Join Paragraphics* (`Ctrl+Shift+J`). Now you have 1 object per line.
- Finally, you can do some optional clean up using *Remove Blank Lines* (`Alt+R`), *Strip Last Character* (`Alt+U`).

Once you got the prepared data, turn to the spreadsheet (we'll be assuming you want to obtain IDs for objects which file size is above or below some threshold):

- Copy lines (some or all, with `Alt+C`) from *Notepad 2e* and paste to the spreadsheet program.
- File size is a numerical field. Depending on your program, it might or might not be one.
  For example, MS Excel automatically removes comma and spaces on paste so nothing needs to be done. *Spread32* doesn't do that but this can be fixed by running *Text to Columns* command on the entire file size column. For others, you can use *Notepad 2e*'s *Replace* from string `, \t` to string `\t` in *Regular expression search* mode – this removes stray commas with spaces.
- Sort the file size column.
- Find where objects with specific file size start. For example, find objects above 1000000 in size (roughly 1 MiB).
- Make a selection in the `RowId` column from the located row to the start or end of the spreadsheet, and copy it.
- Paste to *Notepad 2e* and convert a series of IDs into a *Stagsi* search expression: do *Modify Lines* (`Alt+M`) with the prefix text of `|  /` and then *Join Lines* (`Ctrl+J`), and remove the first pipe.
- The resulting line can be pasted to *Stagsi*'s query input.

Import

Duplicates & Hashes

We often refer to "duplicate" files but what does this mean exactly in *Stagsi*?

To answer this question, we need to briefly explain what a "hash" is (related terms are "cryptographic hash", "digest" and "checksum"). In layman's terms, a "hash" is short for "hash function" that, when fed some data, produces some seemingly random string of a fixed length. The string depends on the data, so that if exactly the same data is fed again, exactly the same string is produced; if the input data has changed even for a little bit, then the string will be different too.

One widely used hash function algorithm is MD5 – short for "Message Digest-5". Contrary to the name, "message" doesn't imply some human-readable text – it works as good for binary data (images, etc.). Here are MD5 hashes of several different strings – as you see, even a mere capitalization resulted in a drastically different hash:

```
string "hello": 5d41402abc4b2a76b9719d911017c592
string "Hello": 8b1a9953c4611296a827abf8c47804d7
string "HELLO": eb61eead90e3b899c6bcbe27ac581660
```

When you import a file, *Stagsi* calculates its MD5 hash and checks if the database already has an entry with the same hash. If it does, the file is considered "duplicate" and usually not imported (depending on the settings).

One obvious advantage of this approach is that it does not depend on file's name, path, modification time or any other properties except the file's contents. If you have 10 Terabytes worth of backups and you want to extract only non-duplicate files, it would take ages to sort by hand, especially if paths and names cannot be trusted. With *Stagsi*, you simply dump the entire archive and let it chew through. In the end, only files with unique contents will remain, and with some scripting you can export such objects preserving the original folder structure (using `OriginalPath` field and/or *Import*'s *Regexp filters*).

Note: of course, one cannot fit all possible variations of data within anyhow short string. There is a chance – called "hash collision" – that two different data would produce identical hashes; however, it is so slim for practical purposes that *Stagsi* ignores it completely, along with the absolute majority of other software.

Zero-Sized Files and the Null Hash

An MD5 hash of an empty "message" (input) is always `d41d8cd98f00b204e9800998ecf8427e`. However, *Stagsi* treats all empty files (0 size) as unique, assuming their `Hash` to be `null` – a special value that does not equal anything, even another `null`.

This allows you to have any number of empty files in your database. Such files are still separate – each has its own tags, thumbnail, `RowId`, etc. but when you import another empty file, it is not marked as duplicate. Use empty files where the data (file) itself has no value, only its "meta-data" (tags, attachments, etc.) is useful.

Note: remember that in SQL to compare something with `NULL` you should use `IS`, not `=` – SqlNULL.

Fuzzy Duplicate Images Detection

Two image files may have different hashes even though their "contents" appears to be identical. It's a

big problem when using software that puts tags into the files themselves (ahem, *Adobe Bridge*, ahem, *Apple iTunes*) because once changed, you cannot tell if two files are the same by merely comparing their contents (what hashing does) and you have to use heavy machinery for this task.

One way to compare two images is using *ImageMagick*'s `compare` utility:

```
compare.exe -verbose -metric mae Image1.png Image2.png diff.png
```

- There are many ways to "compare" images. `-metric` of `mae` simply compares pixel data (colors).
- `diff.png` is a visual difference between `Image1.png` and `Image2.png`.
- `-verbose` makes `compare` output the difference in text as 4 pairs of numbers with 0 meaning "identical".

You can write a script for checking a batch of files, telling you which pictures are the same dispite differences in their raw file data.

Explore Skipped

If an import session didn't process all files, *Stagsi* keeps the import dialog open with the list of files reflecting the skipped items. However, it's not very convenient to work with.

If you want to save such skipped items for later, browse them in *Windows Explorer* or process in some way, use the *Copy* command (`Ctrl+C`). As a result, the clipboard receives two types of data:

- Plain text string – can be pasted into a text editor. Contains 1 file path on a line. For example, if most skipped files appear to come from `F:\Junk` directory, you can check if there are any others by pasting into a blank *Notepad 2e* window, opening *Find* (`Ctrl+F`), setting the search string to `F:\Junk\` and pressing `Ungrep`.
- *Windows Explorer* file list – for pasting into some folder on your system. Sadly, *Explorer* only allows regular *Paste*, not *Paste shortcut* but you can simulate this with other commands.

If you want to explore skipped files but you don't want to copy them entirely, you can create shortcuts using *nircmd*:

- *Paste* (`Ctrl+V`) the skipped file list into a blank *Notepad 2e* window.
- Do *Modify Lines* (`Alt+M`) with the prefix text of `"` and the append text of `"`. As a result, all paths are now wrapped in quotes.
- Edit the text to look like this (nevermind that `(` and `)` brackets end up being on different lines):

  ```
  for %%i in (file list...) do nircmd shortcut %%i E:\Shortcuts
  %%i
  ```

- Save it as a `.bat` file somewhere and run by double clicking in *Windows Explorer* or using the *Launch* command (`Ctrl+L`).
- The `.bat` file is no more needed and can be deleted.

Tags by Wildcard

Often files that we are importing were already organized into some sensible directory structure. For example, photos from this year's birthday might be inside a folder named `Birthday-2019`. If there are hundreds of files, it's nearly impossible (and unnecessary!) to tag them all manually with the same tag.

Stagsi can add tags during import based on a wildcard match. Using our photos example, we can create the following tag hierarchy:

```
+ Personal
  + Birthdays
    + 2018
    + 2019
  + Trips
    + 2019
      + Japan
      + Spain
```

Next, we add a wildcard tagging rule for each set of tags belonging to a particular path. A "wildcard" is a simple string with two special symbols: `*` matching 0 or more characters (any) and `?` matching exactly 1 character (any).

For example:

- `*\Birthday-2019\*` wildcard matches all files which have `Birthday-2019` folder as their parent (immediate or not).
- `*Birthday-201?*` wildcard matches files with "Birthday-201" plus any symbol somewhere in their file name or in one of their parent folders' name, like "My Birthday-2018.jpg".
- `*\Trip*\*` wildcard matches files having a folder beginning with "Trip" like "Trip to Spain".
- `*.jpg` wildcard matches all files with the "jpg" extension. Note that it's usually not needed to tag files by extension because for recognized formats (for which thumbnails can be generated) *Stagsi* adds a special system tag of that format (not extension).
- `R:\*` wildcard matches files on the `R:` disk – perhaps your archive or backup.
- `*` wildcard always matches and can be used to assign a tag to every file being imported. For example, 2 rules can be made: one for `*Birthday*` with "birthday" tag, another for `*` with "all" tag, allowing you to locate all imported images that had no "birthday" tag with "all -birthday". Or you could use the special "Last Imported" system tag if the "all" tag were only temporary (until next import).

Remember: if you want to apply such rules after importing, you can simply re-import the same files (even if they're in another location, they're identified by content hashes) as long as duplicate mode is set to "append".

Tags by Regular Expression (RegExp)

If you are importing files which already have a good directory structure, you may want to preserve some of that structure in form of tags. You can do this with wildcards, but only if the number of tags is small and/or known beforehand – because you cannot create a new tag from a file path dynamically.

This is where regular expressions come into play. They are extremely powerful but also, admittedly,

more cryptic than simple wildcards. The following explanation assumes you have a basic knowledge of their syntax. Online resources can help you in building and validating expressions: uiregex.com, regex101.com, regexr.com.

Let's imagine you are importing a bunch of game screenshots, organized by release year and game name:

```
...\1994\Warcraft: Orcs & Humans\Screenshot0001.png
...\1995\Command & Conquer\...
...\1995\Heroes of Might and Magic\...
...\1995\Need for Speed, The\...
...\1996\Tomb Raider\...
...\1996\Diablo\...
```

You want every file to be tagged with:

- The release year under the `Years` tag – such as "Years/1994" for the first `Warcraft`.
- The game's name under the `Titles` tag – "Titles/Warcraft: Orcs & Humans".
- Additionally, the name must appear under the game's year tag – "Years/1994/Warcraft: Orcs & Humans".

As you notice, out of these tags we only know 2 tags in advance: root `Years` and `Titles` tags. Others need to be extracted from the file names of each file being imported.

We can solve this with a single regular expression:

```
\\(\d{4})\\([^\\]+)\\
```

Let's break it down:

```
\\start matching on any backslash
(backslash is special and so needs to be doubled)
(start a capture group
  \dmatch any digit
  {4}match exactly 4 instances of the preceding term
)end the capture group
\\match a backslash
(start another capture group
  [^\\]match anything ([ ]) that is not (^) a backslash (\\)
  +match 1 or more instances of the preceding term
)end the capture group
\\match a backslash
```

The first file name on our list matches nicely, producing 2 capture groups:

```
...\1994\Warcraft: Orcs & Humans\...
group 1 = 1994
group 2 = Warcraft: Orcs & Humans
```

Now we have groups that hold our desired tag names. How do we assign them?

Each regular expression tagging rule has both an expression and a replacement strings. The expression we have just made. The replacement is simpler – it's a pipe-separated list of tag names (in long form, with parents) where `$n` (like `$1`) are replaced by the `n`'th group's contents. Named groups can be also used: matched with `(?<named>...)` and replaced by `${name}`.

So the second part of the spell is this replacement string:

```
Years/$1 | Titles/$2 | Years/$1/$2
```

...which for our first file name evaluates to:

```
Years/1994 | Titles/Warcraft: Orcs & Humans | Years/1994/Warcraft: Orcs & Humans
```

Stagsi breaks it down into 3 tags, creates tags (parents and the tag itself) as needed and assigns the tags to the file being imported.

Tags From Custom Sources

Naturally, it's impossible for a single program to account for every existing meta-data format and consider all possible users' preferences.

Stagsi does not try to do that. Instead, it provides core support for only a few well-known and popular meta-data formats (such as EXIF used in images) and allows you to supply tags for other situations during import using a simple JSON file containing an array of objects with keys: `File` (imported file's path, relative to the JSON file), `Tags` (array of objects). `Tags` objects have two keys: mandatory `Tag` (array of strings – tag path) and optional `Weight` (defaults to 0). This example assigns two tags: `Artists/"Yuno Nagasaki"` (with -1 weight) and `Year/1990s/1999` (with 0 weight):

```
[
  {
    "File": "..\\music.mp3",
    "Tags": [
      {"Tag": ["Artists", "Yuno Nagasaki"], "Weight": -1},
      {"Tag": ["Year", "1990s", "1999"]}
    ]
  }
]
```

If duplicate `File`'s exist within the same JSON or multiple JSONs, their `Tags` are merged as if they were specified in the same object.

The script below parses ID3v2 tags (found in audio and video files, notably used by iTunes) extracting tags from artist, album, grouping, genre, BPM and comment fields (all comma-separated) into a JSON file. Once it's ran, you perform the usual import making sure to select the JSON file that the script has produced as a "custom meta-data source". If your files are already in *Stagsi*, you can import them again in *Append Tags* mode to merge existing database tags with the extracted ones.

Note: even though the script supports all ID3v2 formats (2.x, 3.x, 4.x), the format itself is very old and programs often use it in non-standard ways so it may fail to extract meta-data from some files,

particularly from the ancient ones.

Import/ID3v2.php • Skip this example (#exYS) • Browse on GitHub • Adjustments (2)

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

$meta = processDir('C:/music');
file_put_contents('c:/tags.json', json_encode($meta));

function nameInfo(array $frames) {
  $res = [];

  $idToTag = ['TCOM' => 'Artist', 'TALB' => 'Album', 'GRP1' =>
'Grouping',
              'COMM' => 'Comment'];

  foreach ($idToTag as $id => $tag) {
    if (!empty($frames[$id])) {
      foreach ($frames[$id] as $s) {
        foreach (preg_split('/\s*,\s*/u', $s) as $s) {
          $res[] = ['Tag' => [$tag, $s]];
          // This script assigns all tags the default weight of 0.
          // You can assign custom weight like so:
          //$res[] = ['Tag' => [$tag, $s], 'Weight' => 20];
        }
      }
    }
  }

  if (!empty($frames['TCON'])) {
    foreach ($frames['TCON'] as $s) {
      $res[] = ['Tag' => ['Genre', $s]];
    }
  }

  if (!empty($frames['TBPM'])) {
    foreach ($frames['TBPM'] as $s) {
      $res[] = ['Tag' => ['BPM', $s]];
    }
  }

  return $res;
}

function processDir($path) {
  $res = [];
  $path = rtrim($path, '\\/').'/';
  foreach (scandir($path) as $file) {
    $full = "$path/$file";
    if (is_file($full)) {
      try {
        $frames = parseFileID3v2($full);
      } catch (NoID3Tag $e) {
        echo "no ID3v2 tag, skipping", PHP_EOL;
        continue;
      } catch (\Throwable $e) {
        echo '!! ', $e->getMessage(), PHP_EOL;
        continue;
      }
      $tags = nameInfo($frames);
      $tags and $res[] = ['File' => $full, 'Tags' => $tags];
```

```
    } elseif (is_dir($full) and $file[0] !== '.') {
      $res = array_merge($res, processDir($full));
    }
  }
  return $res;
}

// https://id3.org/id3v2.4.0-structure
// https://id3.org/id3v2.4.0-frames
function parseFileID3v2($file) {
  echo "== ", preg_replace('![/\\\\]+!u', '\\', $file), " ==",
PHP_EOL;

  $f = fopen($file, 'rb');
  $header = unpack('a3magic/nversion/Cflags/Nlength', fread($f,
3+2+1+4));

  if ($header['magic'] !== 'ID3') {
    throw new NoID3Tag("No ID3v2 tag present.");
  } elseif ($header['version'] > 4<<8) {
    throw new Exception("Invalid ID3v2 tag version
($header[version]).");
  }

  assertNoFlags($header['flags'], 'tag');

  $header['length'] = unsynchInt( $header['length'] );
  echo "ID3v2 tag length: $header[length] bytes", PHP_EOL;

  $framesBuf = fread($f, $header['length']);
  fclose($f);

  if ($header['version'] < 3<<8) {
    $frames = splitID3v22Frames($framesBuf);
  } else {
    $frames = splitID3v2Frames($framesBuf);
  }

  echo "Found ", count($frames), " frames: ", join(' ',
array_map(function ($frame) { return "$frame[id]-
".strlen($frame['data']); }, $frames)), PHP_EOL;
  echo PHP_EOL;

  $usefulFrames = [];

  foreach ($frames as $frame) {
    // Frame types found in iTunes-tagged MP3s:
    //
    // -- 2.x --
    // COM - Comments; iTunes GUI: "comments"
    // TAL - Album/Movie/Show title; iTunes GUI: "album"
    // TBP - BPM (Beats Per Minute); iTunes GUI: "bpm"
    // TCO - Content type; iTunes GUI: "genre"
    // TP1 - Lead artist(s)/Lead performer(s)/Soloist(s)/Performing
group; iTunes GUI: "artist"
    // TP2 - Band/Orchestra/Accompaniment
    // TS2 - ???
    // TSA - ???
    // TT2 - Title/Songname/Content description; iTunes GUI: "song"
    // TYE - Year; iTunes GUI: "year"
    //
    // -- 3.x/4.x --
    // APIC - Attached picture
    // COMM - Comments; iTunes GUI: "comments"
    // GRP1 - ???; iTunes GUI: "grouping"
```

```
    // MCDI - Music CD identifier
    // TALB - Album/Movie/Show title; iTunes GUI: "album"
    // TBPM - BPM (beats per minute); iTunes GUI: "bpm"
    // TCOM - Composer; iTunes GUI: "artist"
    // TCON - Content type; iTunes GUI: "genre"
    // TCOP - Copyright message
    // TENC - Encoded by
    // TIT2 - Title/songname/content description
    // TLEN - Length
    // TMED - Media type
    // TPE1 - Lead performer(s)/Soloist(s)
    // TPE2 - Band/orchestra/accompaniment
    // TPOS - Part of a set
    // TRCK - Track number/Position in set
    // TSSE - Software/Hardware and settings used for encoding
    // TXXX - User defined text information frame
    // TYER - ???
    // WXXX - User defined URL link frame
    $id2to34 = ['COM' => 'COMM', 'TAL' => 'TALB', 'TBP' => 'TBPM',
'TCO' => 'TCON', 'TP1' => 'TCOM'];
    switch ($id = $frame['id']) {
      // 2.x
      default:
        if (!isset($id2to34[$id])) { break; }
        $id = $id2to34[$id];
      // 3.x/4.x
      case 'COMM':
      case 'GRP1':
      case 'TALB':
      case 'TBPM':
      case 'TCOM':
      case 'TCON':
        //echo "> $frame[id]: ", bin2hex($frame['data']), PHP_EOL;
break;
        echo "> $frame[id] ($id):  ";
        assertNoFlags($frame['flags'], 'frame');
        echo join('|', $strings = decodeStrings($frame['data']));
        echo PHP_EOL;
        foreach ($strings as &$ref) {
          $ref = preg_replace('/^\s*|\s*$/u', '', $ref);
        }
        $usefulFrames[$id] = $strings;
        break;
    }
  }

  return $usefulFrames;
}

// 2.x.
// https://id3.org/id3v2-00
function splitID3v22Frames($buf) {
  $frames = [];

  while (strlen($buf)) {
    if ($buf[0] === "\0") { break; }    // padding.
    $header = unpack('a3id/C3length', substr($buf, 0, $hl = 3+3));
    $len = ($header['length1'] << 16) + ($header['length2'] << 8) +
$header['length3'];
    //$len = unsynchInt($len);
    if (strlen($buf) < $hl + $len) {
      throw new Exception("Not enough frame buffer.");
    }
    $frames[] = [
      'id' => $header['id'],
```

```
      'flags' => null,
      'data' => substr($buf, $hl, $len),
    ];
    $buf = substr($buf, $hl + $len);
  }

  return $frames;
}

// 3.x and 4.x.
function splitID3v2Frames($buf) {
  $frames = [];

  while (strlen($buf)) {
    if ($buf[0] === "\0") { break; }    // padding.
    $header = unpack('a4id/Nlength/nflags', substr($buf, 0, $hl =
4+4+2));
    // Contrary to the spec, frame size doesn't seem to be synchsafe
integer,
    // at least in tags produced by iTunes.
    //$len = unsynchInt($header['length']);
    $len = $header['length'];
    if (strlen($buf) < $hl + $len) {
      throw new Exception("Not enough frame buffer.");
    }
    $frames[] = [
      'id' => $header['id'],
      'flags' => $header['flags'],
      'data' => substr($buf, $hl, $len),
    ];
    $buf = substr($buf, $hl + $len);
  }

  return $frames;
}

function unsynchInt($n) {
  $octets = str_split(str_pad(decbin($n), 32, 0, STR_PAD_LEFT), 8);

  foreach ($octets as &$octet) {
    if ($octet[0]) {
      throw new Exception("Synchsafe integer ($n) expected to have
no 8th bits set ($octet).");
    }
    $octet = substr($octet, 1);
  }

  return bindec(join($octets));
}

function decodeStrings($s) {
  $res = [];

  while (strlen($s)) {
    list($string, $tail) = decodeString($s);
    $res[] = $string;
    $s = strlen($tail) ? $s[0].$tail : '';
  }

  return $res;
}

function decodeString($s) {
  switch ($s[0]) {
    case "\0":
```

```
    case "\3":
      $tail = strpos($s, "\0", 1);
      $tail === false and $tail = strlen($s);
      $string = substr($s, 1, $tail - 1);
      $s[0] === "\0" and $string = utf8_encode($string);
      return [$string, substr($s, $tail + 1)];
    case "\1":
    case "\2":
      // It seems NULL byte(s) (for all string encoding types) are
optional and only used if the data does
      // contain several strings, as a separator. For example, found
in COMM only.
      // However, instead of using explode() we still parse strings
one by one
      // because there must be some cases when NULLs are used as
terminator,
      // and then this function would be useful.
      $tail = strlen($s);
      for ($i = 1; isset($s[$i + 1]); $i += 2) {
        if ($s[$i] === "\0" and $s[$i + 1] === "\0") {
          $tail = $i;
          break;
        }
      }
      $enc = $s[0] === "\1" ? 'UTF-16' : 'UTF-16BE';
      return [iconv($enc, 'utf-8', substr($s, 1, $tail - 1)),
substr($s, $tail + 2)];
    default:
      throw new Exception("Unknown string encoding scheme
($s[0]).");
  }
}

function assertNoFlags($n, $type) {
  if ($n) {
    throw new Exception("ID3v2 $type flags not supported ($n).");
  }
}

class NoID3Tag extends Exception { }
```

Multiple File Name Tags

Let's suppose you have a large collection of files where their names (or parts thereof) *are* the actual tags. For example: `sunny sunshine girl landscape public_domain.jpg`. It's not quite convenient to "parse" such a file name into individual tags using the regular technique (see the above section) because the number of tags is unknown, and you'll need to create 1 rule per every possible tag with slightly different regexps.

Note that specifically for these reasons regexp rules allow changing the tag name delimiter, which defaults to `|` (because it's not a valid symbol in file names). To extract tags from our earlier example:

- Set the expression to `\\(.+?)\.\w+$` to extract the file name without extension.
- Set the delimiter to a single space symbol (or to `\s+`).
- Set the replacement string to `\1`.

As a result, first `\1` is replaced with `sunny sunshine girl landscape public_domain` and then the delimiter is applied, making that string a list of 5 tags.

Repairing Corrupted

Sometimes database object files may go bad – power failure, linked files moved to another location, etc. *Stagsi* integrity checks detect this situation and mark such objects with the special *Corrupted* system tag.

If you have proper files somewhere, you can easily repair corrupted objects by importing them. When *Stagsi* detects a file (object) already contained in the database, and that object is corrupted – it replaces most of its properties (including link files and thumbnails) as if the file was imported anew, keeping existing tag assignments.

Change Data Mode

Sometimes you might want to change object import mode after the actual import. For example, files were wrongly imported as links while you wanted them to be stored, or vice versa.

If you want to convert all link-mode objects into "copy", i.e. make a database fully independent of external files, then just run the `/unlink` command-line command.

For other cases, or when you want to selectively "unlink" only specific objects, you can make them "Corrupted" and then re-import with the desired mode (corrupted duplicate files are imported almost as if they were not yet known, just their tags are preserved). One simple way to "corrupt" them is removing their data files from the numbered folders (e.g. `Database\1\1234.jpg` or `1234` for a link) and running `/check`.

Surprising Uses

Audio & Video Bookmarks

By utilizing custom file formats you can realize *Stagsi* in many ways it wasn't meant for in the core – for example, to tag parts of audio and video files. For this one can use M3U playlist files with extensions specific to VLC (a popular media player also called *VideoLAN*).

M3U is a regular text file consisting of blocks, each block addressing a specific playlist entry, which most often is just a file path, but can be preceded by comment lines (starting with `#`) which are often used to add custom information like playback position. Example of such a block with VLC-specific playback position (120 seconds, 2 minutes sharp):

```
#EXTVLCOPT:start-time=120
C:\Users\Elder\Desktop\Indie Game - The Movie.mp4
```

You can create such playlists by hand, via VLC's interface (*Save Playlist* command, `Ctrl+Y`) or with the following VLC extension written in *Lua* which saves a playlist file and a screenshot every time you pause a video (e.g. with `Space`) so that in the end you can import all M3U files and add screenshots as their custom thumbnails. When you *Open* (`Enter`) an M3U file from *Stagsi*, it launches VLC playback exactly at that position.

Import/VLC video bookmarks.lua • Skip this example (#exQL) • Browse on GitHub • Adjustments (5)

```
-- To enable this extension, copy it to VLC's program folder, for
```

```
example:
-- C:\Program Files (x86)\VideoLAN\lua\extensions\
-- ...then (every time VLC starts) enable via the View menu.
--
-- Warning: io.open() doesn't work with UTF-8 but with system-
specific locale.
-- Yet VLC returns playing file path as UTF-8 and have no means to
convert them
-- to CP1251 (or other) but even if it had the conversion would not
be correct
-- for all strings. Therefore Paths with non-Latin symbols will be
mangled on
-- save and all modes will behave as OVERWRITE (this includes video
file names
-- and ..._path options).

local MY_NAME     = 'Stagsi Playlist Maker'

local NONE        = 0
local ADD_NEW     = 1
local APPEND      = 2
local OVERWRITE   = 3

-- After changing options either restart VLC or go to Tools >
-- Plugins and extensions, Active Extensions tab and hit Reload
extensions,
-- then re-enable it via the View menu.
local options = {
  -- Leave empty to use the video's directory. In any case, the file
is prefixed
  -- by the video's base name (without extension) plus a unique
numerical suffix.
  playlist_path   = '',
  -- Warning: make sure to double backslashes!
  --playlist_path   = 'C:\\Stuff\\Save_here',

  -- Like above but for video screenshots.
  snapshot_path   = '',

  -- NONE doesn't write playlist files.
  --
  -- ADD_NEW creates new M3U file.
  --
  -- APPEND adds new lines to the first (existing) M3U file (if no
file exists,
  -- it's created as with ADD_NEW). This allows multiple bookmarks
per one
  -- playlist, which can be viewed and navigated in VLC's View >
Playlist
  -- (Ctrl+L) menu.
  --
  -- OVERWRITE wipes the file if it exists, replacing by the current
(last)
  -- position.
  playlist_mode   = ADD_NEW,

  -- Similar as above but not APPEND (works like ADD_NEW).
  snapshot_mode   = OVERWRITE,

  -- Allowed VLC values - png, jpg, tiff.
  snapshot_format = 'png',
}

function descriptor()
  return {
```

```
    title = MY_NAME,
    version = '1.0',
    license = 'CC0',
    author = 'Soletude',
    description = [[
Whenever a video is paused (often with Space hotkey), saves an M3U
playlist (with current playback position) and/or current video
snapshot for later import into Stagsi.

Open this script in a text editor to configure.
    ]],
    url = 'https://stagsi.com',
    capabilities = {'playing-listener', 'input-listener'},
  }
end

-- Without these two VLC won't recognize the extension.
function activate()
end

function deactivate()
end

-- From lua\modules\common.lua.
local function snapshot()
  local vout = vlc.object.vout()
  if vout then
    -- Saves image to the location and in format specified in VLC
preferences.
    vlc.var.set(vout, 'video-snapshot', nil)
  end
end

-- C:\foo\bar.txt -> C:\foo\
local function split_path(path)
  return path:gsub('[^\\/]+$', ''), path:match('[^\\/]+$')
end

-- path - without format (extension). A prefix will be added by vlc.
-- format - without '.'.
local function snapshot_to(path, format)
  local old = {
    path = vlc.config.get('snapshot-path'),
    prefix = vlc.config.get('snapshot-prefix'),
    format = vlc.config.get('snapshot-format'),
    seq = vlc.config.get('snapshot-sequential'),
  }
  local dir, file = split_path(path)
  -- VLC fails if a path has trailing \, but it also fails if it's a
drive's
  -- root without \. So either C:\... or C:\ but not C:\...\ or C:.
  dir = dir:gsub('[\\/]*$', '')
  if not dir:find('[\\/]') then dir = dir..'\\' end
  vlc.config.set('snapshot-path', dir)
  vlc.config.set('snapshot-prefix', file)
  vlc.config.set('snapshot-format', format)
  -- vlcsnap-00001.png.
  vlc.config.set('snapshot-sequential', true)
  pcall(snapshot)
  vlc.config.set('snapshot-path', old.path or '')
  vlc.config.set('snapshot-prefix', old.prefix or '')
  vlc.config.set('snapshot-format', old.format or '')
  vlc.config.set('snapshot-sequential', old.seq or false)
end
```

```
local function last_file(base, suffix)
  local function exists(file)
    local f = io.open(file)
    return f and (f:close() or true)
  end
  -- VLC's snapshot suffixes are incremental and don't reset to 1
for new
  -- videos. So we see what's the last suffix that exists.
  for i = 1000, 1, -1 do
    local file = ('%s%05d%s'):format(base, i, suffix)
    vlc.msg.dbg(('%s: last_file(%s)'):format(MY_NAME, file))
    if exists(file) then
      return i
    end
  end
end

local function handle_hotkey()
  vlc.msg.info(('%s: handling hotkey'):format(MY_NAME))

  -- file:///C:/%20...
  local video_uri = vlc.input.item():uri()
  local video_file = vlc.strings.decode_uri(video_uri)
    :gsub('^[^/]*/*', '')
    :gsub('/', '\\')

  local video_dir, video_file = split_path(video_file)
  local video_base = video_file:gsub('%.[^.]+$', '')

  vlc.msg.dbg(('%s: video dir %s, file %s, base %s'):format(MY_NAME,
    video_dir, video_file, video_base))

  if options.playlist_mode ~= NONE then
    local base = options.playlist_path:gsub('[\\/]*$', '')
    base = (base == '' and video_dir or base..'\\')..video_base..'-'
    local suffix = last_file(base, '.m3u')
    vlc.msg.dbg(('%s: playlist base %s, suffix %s'):format(MY_NAME,
base, suffix or ''))
    if options.playlist_mode == OVERWRITE then
      if suffix then
        os.remove(('%s%05d.m3u'):format(base, suffix))
      end
    elseif options.playlist_mode == ADD_NEW and suffix then
      suffix = suffix + 1
    end
    -- Convert microseconds to seconds.
    local play_time = vlc.var.get(vlc.object.input(), 'time') /
1000000
    local file = ('%s%05d.m3u'):format(base, suffix or 1)
    local block = '#EXTVLCOPT:start-time=%d\n'..
                  '%s\n'..
                  '\n'
    block = block:format(play_time, video_uri)
    local f = io.open(file, options.playlist_mode == APPEND and 'a'
or 'w')
    f:write(block)
    f:close()
  end

  if options.snapshot_mode ~= NONE then
    local base = options.snapshot_path:gsub('[\\/]*$', '')
    base = (base == '' and video_dir or base..'\\')..video_base..'-'
    local suffix = last_file(base, '.'..options.snapshot_format)
    vlc.msg.dbg(('%s: snapshot base %s, suffix %s'):format(MY_NAME,
base, suffix or ''))
```

```
    if options.snapshot_mode == OVERWRITE then
      if suffix then
        os.remove(('%s%05d.%s'):format(base, suffix,
options.snapshot_format))
      end
    elseif suffix then
      suffix = suffix + 1
    end
    snapshot_to(base, options.snapshot_format)
  end
end

function playing_changed()
  if vlc.playlist.status() == 'paused' then
    local ok, msg = pcall(handle_hotkey)
    if not ok then
      vlc.msg.err(('%s: error: %s'):format(MY_NAME, msg))
    end
  end
end
```

Web Bookmark Manager

If you're the kind of person that keeps hundreds of bookmarks in *Chrome* or *Firefox*, it's likely you are not happy with the way modern browsers manage them – by folders. As such, they have all the limitations of regular file systems vs. tags.

Stagsi may be your key to the kingdom. Much like regular Windows file shortcuts (`.lnk`), there is a format for "Internet shortcuts" (`.url`). In your browser's address bar, try dragging the website's icon and dropping it in a *Windows Explorer* window – a shortcut will be created, that when launched brings you to the target web page.

And since it's a file, it can be imported into *Stagsi*.

Moreover, the format is text-based and can be easily generated. Here's a PHP script that creates `.url` files from *Chrome*'s bookmark export (in HTML format) with proper file names (becoming object title in *Stagsi*):

Import/Chrome bookmarks.php • Skip this example (#exOV) • Browse on GitHub • Adjustments (2)

```
<?php
$file = 'bookmarks_1_2_20.html';
$path = 'output/';

is_dir($path) or mkdir($path, 0777, true);

preg_match_all('/<DT><A HREF="([^"]+)[^>]*>([^<]+)/ui',
               file_get_contents($file), $matches, PREG_SET_ORDER);

foreach ($matches as $match) {
  list(, $url, $title) = $match;
  $title = preg_replace('/[^\pL\pN ]+/u', ' ', $title);
  $title = preg_replace('/^\s*|\s*$/u', '', $title);
  $file = rtrim($path, '\\/')."/$title.url";
  file_put_contents($file, "[InternetShortcut]\r\nURL=$url");
}
```

Supplimenting Regular Folders

Thanks to the linked import mode, you don't have to move all the files into *Stagsi* if you are happy with the traditional file & folder structure but would only like to enhance it with tags without being forced to use *Stagsi* alone (whose *Object Browser* is inherently flat, at least for now).

You can create a database in a special subfolder (say `_Stagsi`) in the root folder of your data, import your files in linked mode and make the links relative (see RelativeLinks). Now, when you "launch" an object from *Stagsi*, it gets redirected to the file 1 level above (and outside) the database – with your files being in their original folders which you can browse as usual with *Windows Explorer* (or maybe with *Total Commander*?).

Given that *Stagsi* is easily Portable and small, you can even copy the entire program into that subfolder and carry it along with your data. To give an idea of the final folder layout:

```
Photos\Family\2019\DCIM4869.jpg          - main data which is being
"enhanced"
Photos\Stagsi.lnk                        - a shortcut for quickly
launching Stagsi
Photos\_Stagsi\Stagsi.exe                - Stagsi program files
Photos\_Stagsi\Database\Stagsi.sqlite    - Stagsi database files
```

Using Windows Shortcuts

With *Stagsi*, a power user must always think beyond what's obvious to open new horizons in organizing one's materials. When "unknown formats" setting is enabled, *Stagsi* accepts not only images but any kind of files – including *Windows Shortcuts* (`.lnk` files). The benefits of this:

- You can tag folders! "Opening" a shortcut object (from *Stagsi*) is equivalent to opening the targeted file, in this case a folder; if it doesn't work for you, then add *explorer.exe* to the list of type associations for `.lnk` files (such entry will appear once you import your first shortcut).
- *Stagsi* is primarily meant for organizing static data, i.e. one that doesn't (often) change. Dynamic content (documents, source code, work-in-progress images, etc.) will constantly fail the integrity checks. One way to work around this is to disable the checks entirely but it's like throwing away a fire extinguisher because it doesn't match the decor. Another is to use per-object attachments but they cannot be tagged and searched. A smarter way if you need all the features except integrity checks is to use shortcuts. Shortcuts themselves remain mostly static unless you move target files around, in which case the path inside the `.lnk` file needs to be changed, changing the file's hash. Files they point to are exempt from integrity checks, without disabling the checks for the entire database.

Shortcuts are easy to create en-masse in Windows Explorer: select all the files you want to be "shortcutted" and *Copy* them (`Ctrl+C`), then right-click on some temporary folder and select *Paste shortcut*. Or drag & drop the files while holding right mouse button (not left) to a temporary folder – on mouse button release, you will get a menu with *Create shortcuts here* command.

Once you have shortcuts, import them in "copy" or "move" mode instead of "linked" to store them within the database folder, avoiding the clutter in your regular directories (like *Desktop* or *My Documents*).

Another Way To Tag Folders

If you want to avoid using shortcuts, you can link directly to folders – the only problem is that *Stagsi*'s import dialog is not currently suited for importing folders.

You can work around this by creating a special file in each folder (e.g. `FolderMarker.txt`), importing those files in linked mode, closing *Stagsi* and doing a *Find & Replace* on the database folder, replacing `\FolderMarker.txt` with a blank string. As a result, `C:\Folder\FolderMarker.txt` becomes `C:\Folder` and "opening" it from *Stagsi* opens a new Windows Explorer window directly.

Using Batch and Other Script Files

With a bit of non-standard thinking, *Stagsi* can be turned into a "mission control"-type program, suitable for managing thousands of commands, each identified by an evolved tag hierarchy.

Let's imagine you have 100 of PCs that you need to maintain in some way. Each PC may have a set of batch files to perform tasks such as "reboot", "install updates", "make screenshot", etc. Each batch file may have at least two tags: the PC's name (under "PCs" root tag) and task name (under "Tasks" root tag). Whenever you want to "install updates" to a particular PC, you search for "PC/Marys" or to "Tasks/Update", or even to "PC/Marys Tasks/Update" if you have a lot of PCs and tasks. Select your batch file in the list and run it with `Enter` (a hotkey to the *Open* command).

Multiple commands can be ran the same way – with `Enter`, but if you want more control, then you can export them to a temporary folder (by dragging thumbnails from *Stagsi* to *Windows Explorer* or using `Ctrl+S` – *Export* command).

Remember: *Stagsi* has its own type associations (per database), which default to the system ones but can be change separately. This way, if outside of *Stagsi* your `.ps1` files are associated with *VS Code* for faster editing, within *Stagsi* you can associate them with the *PowerShell* interpreter directly for instant run (you can still edit the file within using the *Show in Explorer* command, `Alt+Enter`).

You can even create new associations that do not exist outside of *Stagsi*. Say, if you're a heavy *nircmd* user then you can write your scripts with *.nir* extension and associate it to *nircmd.exe script* within *Stagsi*, without affecting global Windows associations. If you put *nircmd.exe* in the database folder, then such association becomes portable and will work on any system where *Stagsi* can be ran.

Command-Line

Basic *Stagsi* command-line syntax is:

```
Stagsi.exe [/ro] [/multi] [/sql [log.sql]] [[data-path] [/op...]]
```

By default, *Stagsi* launches in read/write, single instance mode, with `data-path` read from `DataPath` key in `Bootstrap.json` in the program's folder (which defaults to `%APPDATA%\Soletude\Stagsi\Database`).

Arguments after `data-path` are operation-specific arguments, where the first (`/op`) is the operation's identifier.

Some operations (`/search`, `/hash`, `/tags`, `/import`) are "combining", i.e. when called without

`/multi` and the same database (`data-path`) is already opened in *Stagsi*, that process is signaled to perform that action (e.g. open the *Import* dialog). Non-combining operations like `/rethumb` only activate the running process.

Read-Only

Sometimes you want to browse the database without making any changes. The `/ro` switch enables read-only mode, protecting against accidental modifications. Other consequences of this mode:

- `/multi` is enabled because to enforce single-instance mode, a lock file needs to be created in `data-path` (making this mode no more "read-only").
- Searches may run slightly slower because search caching (compiled assemblies in `DataPath\Temp`) is disabled.

`/ro` is enabled automatically if the database file is not writeable (tested on launch). This allows running *Stagsi* from CD-ROM, network shares, etc.

Multi-Instance Mode

By default, if there is a *Stagsi* process running on a given database (`data-path`), attempt to run another process on the same database will fail, bringing the already running one to the foreground. Generally, this is what you want – keep *Stagsi* running in background (to preserve its caches).

By passing `/multi` you disable the single-instance check. However, do not run multiple processes that perform modifications – *Stagsi* reads most data into the memory on launch and doesn't refresh it; when two processes change data, they may create inconsistencies. Because of this, all but one process must be ran as `/ro` (read-only), or at least ensure that they do not perform conflicting actions (there's no definite list of such actions though).

Internally, when a process tries to open a database (with or without `/multi`), it creates a mutex and a lock file (`DataPath\Lock`). The lock file holds the ID (PID) of the first process that has opened this database. If the mutex could not be created, the process exits unless `/multi` was given. If the database wasn't open before, then `/multi` has no effect and the mutex and lock file are created anyway.

SQL Log

If `/sql` is given, *Stagsi* writes all executed non-`SELECT` SQL statements to a text file. If there is a file argument after `/sql` (must not begin with `/` or be empty `""`) then that file is appended to, else a unique `.sql` file is created directly under `DataPath`.

The text file contains valid SQL instructions (alike to a database dump) and can be "applied" to an older `Stagsi.sqlite` database using any tool – SQLite. `/sql` can be used as a log journal for manual review of the database changes, as a differential backup or as a "patch" for a database on another machine (e.g. in a basic master-slave setup).

Data-Path

If this argument is missing, it's set according to `DataPath` key in `Bootstrap.json` in the program's folder.

If it's given, it may be one of the following (can be conveniently used with drag & drop of a file or folder

in *Windows Explorer* onto *Stagsi*'s icon):

- Path to `Stagsi.sqlite` file.
- Path to an existing directory. Assumed to be a database, with missing files (*Stagsi.sqlite*, *Settings.json* and others) created automatically.
- A non-existing path. Assumed to be a path to a database directory; non-existing components are created as directories. For example: `E:\Data\Stagsi` – if it doesn't exist, `Data` directory is created, *Stagsi* inside it, then `Stagsi.sqlite` and others are created inside *Stagsi*.

/search

This command opens a new tab with the given search query. Apostrophes `'` are replaced by quotes `"` to help working around weird Windows command-line quoting quirks.

This command can be used to facilitate a more specialized search in a separate program and send found results back to *Stagsi* in the form of "ID" search query: `/22 /34 /123 /201 ...` (QueryId). If *Stagsi* is already running, normally this would activate it and search, otherwise it will be started and search triggered.

The sample repository contains a *C# + WinForms* program using the official SQLite bindings to provide an "extended" search dialog which filters by most `Objects` table fields such as file size and hash. *WinForms* is used because it provides advanced text controls (spinners and data pickers) out of the box, but it's fairly easy to rewrite the sample using WPF or UWP.

If you want to close all tabs first, you can `taskkill` *Stagsi*, delete `Queries.json` (this is where tab state is stored) and then run the `/search` commands.

/nmh

Google Chrome allows its addons to communicate with programs via a "native messaging host" protocol. By default *Stagsi* is coming with `NMH.exe` which implements it, reading messages and writing responses in an infinite loop that can be broken with `Ctrl+C` or `Ctrl+Z` (EOF).

See *Chrome*'s documentation for details: developer.chrome.com/extensions/nativeMe….

The included `NMH.exe` lets you do this:

- Spawn a new *Stagsi* process by combining arguments of `NMH.exe` with the request's `args` array. For example: `NMH.exe /ro` will always spawn processes in read-only mode. Some *Chrome*-specific arguments (like `chrome-extension://`) are ignored.
- Save files to the database's `Temp` (DataTemp) folder, since doing that using WebExtension API is just too hard. Files get random unique names plus extensions that you specify.

Request object keys: `do` (required, one of `spawn` or `save`). For `spawn`, also these: `args` (required, an array of strings, starting from `/op`), `wait` (optional, if the response should be delayed until the process exits). For `save`: `ext` (required, new file's extension), `data` (base64-encoded string).

Response object keys: `error` (boolean or string if have detailed error info). For `spawn`, also these: `code` (integer, exit code, only if `wait` and no `error`), `clipboard` (textual clipboard contents, only if `wait` and no `error`; useful for `/pick`). For `save`: `file` (absolute file path).

Below is a simple *Google Chrome* add-on that runs the search in *Stagsi* from the omnibox with "stg" prefix (example: "stg landscape | nature" runs `Stagsi.exe /search "landscape | nature"`), as well as from the toolbar button, using current page's selection.

- First, we need to generate a private/public key pair for this add-on because the add-on's ID need to be known in order to be used within NMH. One simple way of doing this is creating an empty add-on (manifest below), going to *Extensions*, *Pack extension*, then dragging and dropping the created `.crx` file into the *Extensions* tab, going to `%APPDATA%\Local\Google\Chrome\User Data\Default\Extensions`, locating your just installed add-on there and copying the public part from its `manifest.json` (the private part was saved by the *Pack* command earlier).

```
{
  "name": "Empty add-on",
  "version": "1.0",
  "manifest_version": 2
}
```

- Create a *Native Messaging Host* manifest from this template saved in `C:\Stagsi-addon\nmh.json`:

{} NMH/nmh-manifest.json • Skip this example (#exOA) • Browse on GitHub • Adjustments (4)

```
{
  "name": "ca.soletude.stagsi.my-addon",
  "description": "Stagsi NMH for My Addon",
  "path": "C:\\Program Files\\Stagsi\\NMH.exe",
  "type": "stdio",
  "allowed_origins": [
    "chrome-extension://diikokbialbifgbobhohkjgaelcohcbo/"
  ]
}
```

Note: `path` doesn't allow arguments so if you want to pass them you have to create and register a "proxy" shortcut (`.lnk` only, `.bat` doesn't work) that calls `"C:\Stagsi\NMH.exe" /ro "C:\My DB" ...`.

- Register the NMH manifest, in the current user's registry section (or in `HKLM` for all users):

```
reg add
HKCU\Software\Google\Chrome\NativeMessagingHosts\ca.soletude.stagsi.my-
addon /ve /d "C:\Stagsi-addon\nmh.json" /f
```

- Create a manifest for our add-on:

{} NMH/addon-manifest.json • Skip this example (#exPB) • Browse on GitHub • Adjustments (6)

```
{
  "name": "My Search in Stagsi",
  "version": "1.0",
  "manifest_version": 2,
  "description": "Runs Stagsi /search from page selection and
omnibox.",
```

```
  "omnibox": {"keyword": "stg"},
  "background": {
    "scripts": ["background.js"],
    "persistent": false
  },
  "browser_action": {},
  "content_scripts": [
    {
      "matches": ["<all_urls>"],
      "js": ["content.js"],
      "run_at": "document_start",
      "all_frames": true
    }
  ],
  "icons": {
    "16": "icon16.png"
  },
  "permissions": [
    "nativeMessaging"
  ],
  // In this example we assume diikokbialbifgbobhohkjgaelcohcbo
ID.
  "key": "..."
}
```

- Finally, write the scripts. The add-on is split into two parts: a background service worker (handles the omnibox) and a content script (retrieves selection from the active tab).

{} NMH/background.js • Skip this example (#exFH) • Browse on GitHub • Adjustments (1)

```
function exec(args) {
  chrome.runtime.sendNativeMessage(
    'ca.soletude.stagsi.my-addon',
    {do: "spawn", args},
    function (resp) {
      if (chrome.runtime.lastError || !resp || resp.error) {
        alert('Unable to contact Stagsi: ' +
chrome.runtime.lastError)
      }
    }
  )
}

chrome.browserAction.onClicked.addListener(function () {
  chrome.tabs.query(
    {
      active: true,
      windowId: chrome.windows.WINDOW_ID_CURRENT
    },
    function (tab) {
      chrome.tabs.sendMessage(tab[0].id, {}, function (resp) {
        var args = resp ? ['/search', resp] : []
        exec({args})
      })
    }
  )
})

chrome.omnibox.onInputEntered.addListener(function (text) {
  exec({args: ['/search', text]})
})
```

NMH/content.js • Skip this example (#exRD) • Browse on GitHub

```
chrome.extension.onMessage.addListener(function (req, sender,
send) {
  send(window.getSelection().toString())
})
```

Troubleshooting

Use the below C version to see if the NMH program is launched, that it receives expected messages and your addon fetches its responses. This rules out various manifest-related errors.

If *Chrome* reports that `Specified native messaging host not found` then the manifest file is not registered (in the Registry) or that it's referencing a non-existing program (`path`). If it says `Error when communicating with the native messaging host` then the host exists but it can't be ran (e.g. it's a `.bat` file, not `.exe` or `.lnk`) or that its response is empty (EOF) or bad (e.g. invalid length `DWORD`).

C Implementation

Google Chrome is quite peculiar about the nature of the host and doesn't seem to call any script files (`.bat`, `.php`, etc.). For debug purposes you can use this small C/C++ program to log all input to a text file and to `stdout` so your addon should receive exactly what it has sent:

NMH/nmh-stub.c • Skip this example (#exMA) • Browse on GitHub • Adjustments (1)

```
#define _CRT_SECURE_NO_WARNINGS
#include <stdio.h>
#include <stdlib.h>
#include <windows.h>

int main(void) {
  FILE *debug = fopen("nmh.log", "w");
  fprintf(debug, "CL: %s\n", GetCommandLine());
  setbuf(debug, NULL);
  setbuf(stdin, NULL);
  setbuf(stdout, NULL);
  while (1) {
    DWORD len;
    if (fread(&len, 1, 4, stdin) != 4) { return 1; }
    fprintf(debug, "in: %u\n", len);
    char *buf = (char*) malloc(len + 1);
    if (fread(buf, 1, len, stdin) != len) { return 2; }
    buf[len] = 0;
    fprintf(debug, "  %s\n", buf);
    fwrite(&len, 1, 4, stdout);
    fwrite(buf, 1, len, stdout);
  }
}
```

Here's a compiled version (2 KiB): github.com/Soletude/Stagsi-Coo….

/hash

This command allows locating one or more files in a database. For example, to add a new command to

Windows Explorer's context menu that displays the selected file in *Stagsi* (if it's part of the database) import this `.reg` file:

Misc/Find in Stagsi.reg • Skip this example (#exJN) • Browse on GitHub • Adjustments (1)

```
Windows Registry Editor Version 5.00
[HKEY_CURRENT_USER\SOFTWARE\Classes\*\shell\Find in Stagsi\command]
@="\"C:\\Program Files (x86)\\Stagsi\\Stagsi.exe\" /hash \"%1\""
```

If you'd rather have it under the *Send To* menu then add a shortcut (`.lnk`) to `%USERPROFILE%\SendTo` folder, which on Windows 10 resolves to `C:\Users\your_user_name\AppData\Roaming\Microsoft\Windows\SendTo`. Using *nircmd*:

```
nircmd shortcut "Stagsi.exe" ~$sys.userprofile$\SendTo "Find in
Stagsi" /hash
```

Let's say you're using *Stagsi* as a "cataloguing" replacement for WinAMP – you use the latter to play media files but you queue and manage them using *Stagsi* using a Tools script like so:

Tools/Selection to &WinAMP.php • Skip this example (#exTU) • Browse on GitHub • Adjustments (1)

```
<?php
$data = json_decode(strstr(file_get_contents(getenv('SELECTION')),
'{'));
$cl = ['C:/Program Files/winamp/winamp.exe'];
foreach ($data->Objects as $object) {
  $cl[] = $object->FilePath;
}
system(join(' ', array_map('escapeshellarg', $cl)));
```

If you want to change tags (e.g. rating) of the currently playing track, use WinAMP's *Explore item(s) folder* (`Ctrl+F` in *Playlist Editor*) to locate its file and then use *Explorer*'s *Find in Stagsi* command set up above to select it in *Stagsi*.

/import

This command displays the *Import* dialog with last import settings and file list read from the command-line (recursively adding files from directories).

Let's imagine you are a NES addict and you play games in *FCEUX* (an open source NES emulator). *FCEUX* supports scripting in *Lua* language. Here's how you can bind the `Z` key for taking screenshot and pushing it to *Stagsi* in one keystroke:

Import/FCEUX screenshots.lua • Skip this example (#exLJ) • Browse on GitHub • Adjustments (3)

```
local temp = os.getenv('TEMP')..'\\stagsi-fceux'
local last = 0

while true do
  if input.get().Z and os.difftime(os.time(), last) >= 1 then
    gui.savescreenshotas(temp)
    emu.frameadvance()
```

```
    os.execute('"C:\\Program Files\\Stagsi\\Stagsi.exe" /import
'..temp..'')
    os.remove(temp)
    last = os.time()
  else
    emu.frameadvance()
  end
end
```

/pick

This command provides a very simple interface for using *Stagsi* as a kind of "open file" dialog.

With `/search` you can open a new tab (as in the `/op` of `/search`).

With `/force` you can add hidden criteria using the same query syntax (`/force` is ignored on error). Using this option alone is useful to limiting a *Stagsi* instance to a specific subset of tags.

With `/count` you can specify how many results you require to be returned (`min` defaults to 1, `max` – to "infinity").

Stagsi exits with code 0 if user has picked object(s) meeting these criteria (`/force` and `/count`) – in this case clipboard is changed as if the *Copy* command was used allowing the caller to retrieve picked files' paths, description as a JSON or first valid image's data.

Below is a simple *C* program (can be compiled with *MSVS* or *gcc* under *cygwin*) that asks *Stagsi* for a wallpaper image and then sets it to all displays obtaining its path from *Windows Explorer*'s list of "pasted" files:

Misc/Pick wallpaper.c • Skip this example (#exPO) • Browse on GitHub • Adjustments (2)

```
#include <stdio.h>
#include <windows.h>

int main(int argc, char** argv) {
  // Attention: CreateProcessW (even if it's not used in this
example)
  // requires lpCommandLine to be writeable.
  char cl[MAX_PATH];
  snprintf(cl, MAX_PATH,
    "\"%s\" /pick /search \"%s\" /force \"%s\" /count 1 1",
    "C:\\Program Files\\Stagsi\\Stagsi.exe",
    "Wallpapers",
    "(PNG | JPG | GIF | BMP)");

  STARTUPINFO si = {0};
  PROCESS_INFORMATION pi = {0};
  BOOL processOK = CreateProcess(NULL, cl, NULL, NULL, FALSE, 0,
NULL, NULL, &si, &pi);
  WaitForSingleObject(pi.hProcess, INFINITE);

  DWORD code;
  if (!GetExitCodeProcess(pi.hProcess, &code)) {
    code = (DWORD) -1;
  }

  if (code == 0) {
    // Successfully picked an image.
    OpenClipboard(NULL);
```

```
    HANDLE drop = GetClipboardData(CF_HDROP);
    wchar_t fn[MAX_PATH];

    if (drop != NULL && DragQueryFileW(drop, 0, fn, MAX_PATH - 1)) {
      BOOL wallpaperOK = SystemParametersInfoW(SPI_SETDESKWALLPAPER,
0, fn,
        SPIF_UPDATEINIFILE | SPIF_SENDCHANGE);
      if (wallpaperOK) {
        CloseClipboard();
        return 0;
      }
    }

    CloseClipboard();
  }

  return 1;
}
```

Maintenance

There are a number of service operations that you normally don't need to call in a default setting.

/op	Meaning	Affected system tag	When to call
/check	Calculate hashes for object data on disk and compare with ones in the database.	Corrupted	When you have disabled BackgroundChecks setting and/or want to manually trigger the integrity check.
/rehash	Abandons object data hashes (and file sizes) from the database in favour of existing data on disk. Hashes are preserved for objects with a missing or a zero-sized data file, and for ones whose new hash is already contained in the database (i.e. another object has the same hash).	Corrupted	When object data in *Data path* (or linked target files) was changed and you are sure the new state should be perceived as valid from now on.
/rerand	Randomize all database objects, effectively changing the Random field.	None	When you want the *Order by Random* sort mode to generate new order.
/rethumb	Regenerates thumbnails according to the current state of the database and files.	Custom Thumbnail	When manual changes were made to Thumb... fields in the database or to t, u, c-prefixed files in the *Data path*.

`/unlink`	Replace all objects in linked mode with non-linked mode, copying targeted file data into the database.	None	When you want a database to become self-contained, not relying on external data files.

System Scheduler

You can schedule running maintenance tasks with the standard *Task Scheduler* of *Windows*. It's a powerful tool with a friendly GUI.

Below is a sample `.xml` task configuration (using the GUI or the command-line) that runs `/dbcheck`, `/rerand` and `/check` every 3 days at night (2 AM) – longer-running commands last so as not to cause others to be never ran since they have less chances of finishing.

Misc/Maintenance scheduler.xml • Skip this example (#exUB) • Browse on GitHub • Adjustments (5)

```
<?xml version="1.0" encoding="UTF-16"?>
<Task version="1.2"
xmlns="http://schemas.microsoft.com/windows/2004/02/mit/task">
  <Triggers>
    <CalendarTrigger>
      <StartBoundary>2019-01-01T02:00:00</StartBoundary>
      <Enabled>true</Enabled>
      <ScheduleByDay>
        <DaysInterval>3</DaysInterval>
      </ScheduleByDay>
    </CalendarTrigger>
  </Triggers>
  <Settings>
    <WakeToRun>true</WakeToRun>
  </Settings>
  <Actions Context="Author">
    <Exec>
      <Command>"C:\Program Files\Stagsi\Stagsi.exe"</Command>
      <Arguments>/dbcheck</Arguments>
    </Exec>
    <Exec>
      <Command>"C:\Program Files\Stagsi\Stagsi.exe"</Command>
      <Arguments>/rerand</Arguments>
    </Exec>
    <Exec>
      <Command>"C:\Program Files\Stagsi\Stagsi.exe"</Command>
      <Arguments>/check</Arguments>
    </Exec>
  </Actions>
</Task>
```

The configuration can be imported via the GUI or by this command, and it will process the database of the importing user:

```
schtasks /create /xml "stagsi.xml" /tn "Stagsi Maintenance"
```

Partial /rehash

As of now maintenance commands always process the entire database and cannot be limited to processing specific objects only. If your database is large and its files change often (maybe because you have imported WindowsLNK), you might be in need of updating hashes of selected objects only.

This script accepts the list of objects' `RowId`'s (conveniently obtained using the *Property* panel's `RowId` input's *Copy* button) and updates their hashes:

Database/rehash.php • Skip this example (#exXU) • Browse on GitHub

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

$self = array_shift($argv);
$dbPath = array_shift($argv);
$ids = $argv;

if (!is_file($dbPath) or !$ids) {
  echo "Usage: php ", basename($self), " stagsi.sqlite id [id id
...]", PHP_EOL;
  exit(1);
}

chdir(dirname($dbPath));
$settings = json_decode(file_get_contents('Settings.json'));
$db = new PDO("sqlite:$dbPath");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$getStmt = $db->prepare("SELECT * FROM Objects WHERE rowid = ?");
$setStmt = $db->prepare("UPDATE Objects SET Hash = ? WHERE rowid =
?");

foreach ($ids as $id) {
  $id = (int) $id;

  $getStmt->bindValue(1, $id);
  $getStmt->execute();
  $row = $getStmt->fetchObject();

  if (!$row) {
    echo "$id: no such row", PHP_EOL;
    continue;
  }

  $base = floor($id / $settings->FolderSize)."/$id";
  $file = is_file($base) ? file_get_contents($base) : "$base.$row-
>Format";

  if (!is_file($file)) {
    echo "$id: no data file: $file", PHP_EOL;
    continue;
  }

  $setStmt->bindValue(1, $hash = md5_file($file));
  $setStmt->bindValue(2, $id);
  $setStmt->execute();

  echo $hash === $row->Hash ? "$id: unchanged" : "$id: updated",
PHP_EOL;
}
```

The following alternative version does the same but is called from the Tools menu and updates hashes of the objects selected in *Object Browser* (*Stagsi* is closed and reopened on completion). The script's file name must start with `Selection`.

Tools/Selection rehash.php • Skip this example (#exLZ) • Browse on GitHub

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

$selection =
json_decode(strstr(file_get_contents(getenv('selection')), '{'));
system('taskkill /pid '.escapeshellarg(getenv('pid')));

for ($i = 0; $i < 50; ++$i) {
  $line = system('tasklist /fi "pid eq
'.escapeshellarg(getenv('pid')).'" /fo csv');
  if (strpos($line, '","') === false) {
    $i = -1;
    break;
  }
  usleep(100000);
}

if ($i >= 0) {
  echo "cannot terminate Stagsi", PHP_EOL;
  exit;
}

$db = new PDO("sqlite:Stagsi.sqlite");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$setStmt = $db->prepare("UPDATE Objects SET Hash = ? WHERE rowid =
?");

foreach ($selection->Objects as $obj) {
  $id = (int) $obj->RowId;

  if (is_file($obj->FilePath)) {
    $setStmt->bindValue(1, $hash = md5_file($obj->FilePath));
    $setStmt->bindValue(2, $id);
    $setStmt->execute();
  }
}

system('start "Stagsi" '.escapeshellarg(getenv('stagsi')));
```

Note: these scripts do not remove the `Corrupted` system tag. To do so, view "corrupted" object(s) in *Stagsi* and it will soon correct the state of this tag in background according to the new hash.

Other Partial Commands

We have PartialRehash a few useful scripts for performing partial `/rehash`. For other commands, you can either roll your own script or use this trick:

- Close *Stagsi*.
- Open your database's folder.

- Move all numbered folders except the ones with the files you want to "re-something".
- Run `Stagsi.exe /re-something`. Because most commands ignore broken objects (with no data files), it will essentially process only objects in those numbered folders you've left (e.g. with `RowId`'s 1000-1999).
- Once it's finished, move the numbered folders back into the database's folder.

Settings

Not all *Stagsi* settings are accessible via the GUI. You can change everything it has via the `Settings.json` file using a text editor like *Windows Notepad* (while the program is not running).

The following settings exist:

Name	Default	Meaning	Effect	Change implications
`BackgroundChecks`	`True`	If enabled, recently viewed objects are added to a queue. When *Stagsi* is idling, it calculates hashes for objects in this queue and compares to the `Hash` field in the database, adding/removing the `Corrupted` system tag accordingly.	If disabled, browsing might be more performant on very low-end hardware (even though it's designed to be non-intrusive and working even over USB) but you won't be notified of data corruptions unless you trigger the check manually (`/check`).	None.
`CacheImagesInMemory`	`True`	Preloads thumbnails you are likely to see in advance from disk.	If enabled, the disk experiences higher activity but scrolling is smoother.	None.
`FolderSize`	1000	An approximate number of files per each folder. The real maximum is about 4 times that, if every single object has a standard, user and crop thumbnails.	If made too small or too large, everything may be slowed down.	Will fail to locate already imported objects and their thumbnails until they are manually moved around to respective folders (there is no built-in command for that).

`FreeSpaceThreshold`	100 MiB	Affects import and other operations that may write a large number of data to disk.	*Stagsi* makes a guess at the beginning of such operation and bails out if the remaining space of the disk where the database is located is less than this value. If set to 0, no such check happens but import may fail in the middle with your system running completely out of space.	None.
`ImportUnknownFormats`	`True`	Allows importing every type of file, including files for which *Stagsi* cannot generate thumbnails automatically.	If enabled, files of "unknown" formats still work but you'll see placeholder images instead of actual thumbnails, unless you add one manually.	None.
`LargeFileThreshold`	10 MiB	Used in two operations: during import in *Auto* mode (files below this value are copied, others – linked to) and hash calculation (only last – note: last! – this many bytes of a file are hashed).	Affects import in the explained way.	May render already imported objects' hashes invalid, mistakenly marking them with `Corrupted`. This can be fixed by running `/rehash` (assuming no object has a truly corrupted data).
`MaxImageCacheSize`	1.5 GiB	An approximate maximum amount of RAM that *Stagsi* may use per process. Note that *Stagsi* is a Net program and this limit isn't guaranteed to be very accurate.	Low value may worsen the performance. Overly high value may cause your system to be unstable due to lack of memory.	None.

`MaxThumbWidth`	1000 pixels	When *Stagsi* generates a thumbnail (the `t` file prefix) and it's above this dimensions, it's scaled down.	Directly linked to your zoom level. If your thumbnails are larger than this value, they will look blurry. But making this setting very large will cause high memory usage and poor performance. Should be set to the maximum (or slightly less) thumbnail dimension that you expect to ever use during browsing.	Will be in effect for newly imported objects or after `/rethumb`.
`NoThumbThreshold`	20 MiB (200 MiB prior to 1802)	Max file size for known formats for which thumbnails are generated and metadata[1] is extracted.	Files larger than this will get no automatic thumbnail and tags[1]. Setting it too high may cause out of memory errors during import. [1] Since *Stagsi* version 1802, this only affects metadata detection. Thumbnails are now generated for files of any size.	None.
`ScaleStep`	`1.5`	The factor by which thumbnail size changes when using the buttons and hotkeys in the browser.	Low value – more gradual zooming, requiring more clicks to get to a higher/lower zoom level.	None.
`ThumbFormat`	`JPG`	One of `png` or `jpg`. Specifies the format for standard, user and cropped thumbnails.	PNG is best for sharp images and pixel art but has larger file size (leading to higher memory usage and potentially lower performance). JPEG is best for photos but produces blurry images for low-resolution images like pixel art.	No thumbnails will be visible for already imported objects until `/rethumb` is called.

`ThumbQuality`	80	Only if `ThumbFormat` is `jpg`. Sets image quality in % where 100 (%) is the best possible quality – marginally better than 95 or even 90 but much larger in size.	Low value – low picture quality, small thumbnail files (less memory usage, better performance).	Existing thumbnails will use old quality until `/rethumb` is called.
`Language`		User interface language as a two-letter code (e.g. `ru` for Russian). Empty value uses the user's display language. If the requested localization is unavailable – English is used.	Affects most displayed texts. The cookbook may not be available in all supported languages.	Some texts are used during database initialization only (e.g. system tag names) and do not change with this setting.

License Key

Stagsi may operate in two basic modes: limited (freeware) and unlimited (licensed). Effective license key is read from two locations, in this order:

- `%APPDATA%\Soletude\Stagsi\Stagsi.lic` – suitable for default installations (each PC user having his own license regardless of *Stagsi* installation folder).
- `Stagsi.lic` in the folder of `Stagsi.exe` – suitable for portable installations (applied to all PC users running *Stagsi* from the same folder).

These are the only locations where license info is stored. To activate a license put a key file there and restart *Stagsi* (if it was running); there is no online activation or other requirements. If no valid key files exist then the `Free` edition is made effective.

License key is a data packet signed by one of the *Soletude*'s RSA keys, public parts of which you can obtain from here: go.soletude.ca/stagsi/license/rsa (they are also hardcoded into *Stagsi*). To decode:

- Ensure the first byte is `0x41` (ASCII letter `A`).
- Discard it and decode the rest using base64.
- First 256 bytes are the signature of the following (257-…) bytes.
- Bytes from 257 onward constitute a compressed GZ/DEFLATE stream (can be read with .Net's `DeflateStream` or PHP's `gzinflate()` but not *nix' `gunzip`).
- Separate the uncompressed data by NULL bytes (`0x00`) and further separate each part into key/value pairs (keys end with an ASCII space, `0x20`).

These keys are known:

- `E` – license type symbol, such as `P` for "Professional".
- `N` – name of the licensee.
- `I` – some informational text displayed in the UI.
- `D` – a pair of space-separated dates telling the earliest and the latest *Stagsi* versions for which this license key works. Current *Stagsi*'s release date is displayed in the *About* box.
- `M` – machine identifier, for licenses bound to a particular hardware (not user).

Other Tips

- *Stagsi* creates missing database files automatically. This means that to create a new database you simply need to create a new folder and open it in *Stagsi*.
- Zoom level (thumbnail size) is a global setting, not per-tab – unlike sort mode and others. This is because it's directly related to thumbnail cache (which is also global and only one per *Stagsi* process); allowing different zoom levels per tab would cause the memory usage to skyrocket (different cache per each different zoom level). However, you can open multiple *Stagsi* processes on the same database (even though all but one have to be in read-only mode).

Exporting to Spreadsheet

Stagsi can be used as an intermediate tagging/management utility – it takes some data, you tag it, then give results to some other tool. Luckily, open formats make it very easy both to feed and to pull the data.

Here is a single SQL query that produces a table of 1 row per object, with the information like its title, hash and list of tags (only titles, no parents):

```
SELECT o.Title, o.Hash, o.FileSize, o.Format,
       GROUP_CONCAT(t.Title, '"')
  FROM Objects o
  JOIN ObjectTags ot
    ON ot.ObjectRowId = o.RowId
  JOIN Tags t
    ON t.RowId = ot.TagRowId
 GROUP BY o.RowId
```

By using it with *sqlite3* you can create a file that can be opened in any program understanding CSV ("comma-separated values") format – note the triple `"` which is *cmd.exe*'s way to escape the quotes:

Export/Spreadsheet.bat • Skip this example (#exDT) • Browse on GitHub • Adjustments (1)

```
sqlite3.exe -csv Stagsi.sqlite "SELECT o.Title, o.Hash, o.FileSize,
o.Format, GROUP_CONCAT(t.Title, '""') FROM Objects o JOIN
ObjectTags ot ON ot.ObjectRowId = o.RowId JOIN Tags t ON t.RowId =
ot.TagRowId GROUP BY o.RowId"
```

For *MS Excel* you have to use `;` instead of `,` as the separator:

Export/Excel.bat • Skip this example (#exQA) • Browse on GitHub • Adjustments (2)

```
sqlite3.exe -csv -separator ";" Stagsi.sqlite "SELECT o.Title,
o.Hash, o.FileSize, o.Format, GROUP_CONCAT(t.Title, '""') FROM
Objects o JOIN ObjectTags ot ON ot.ObjectRowId = o.RowId JOIN Tags t
ON t.RowId = ot.TagRowId GROUP BY o.RowId"
```

We are using quotes (") to separate tag names because quotes is the only symbol not valid in a tag title and it cannot be escaped.

Exporting with Keywords

Unlike other popular tools like *Adobe Bridge* and *Apple iTunes*, *Stagsi* does not store the meta-data within the file (image or other) itself. This means if you open the file outside of *Stagsi* (or copy it by means of the *Export*, `Ctrl+S` command) you don't see any of your tags.

There are many reasons against modifying files in-place, one being that *Stagsi*'s tag hierarchy is complex and it's unclear how to fit it into plain-text keyword fields that not even some formats have (e.g. ID3 audio tags don't) – or that they properly support Unicode symbols (most of meta-data formats are ancient).

However, in some cases you wish to export images with a simplified tag layout and/or fill other meta-data properties. Here's a PHP script that is using built-in `iptcembed()` function to add keywords ands authorship (identified by children of a specific tag) to JPEG images (only JPEG) that will show up in *Windows Explorer*'s file properties:

Export/With keywords.php • Skip this example (#exMS) • Browse on GitHub • Adjustments (3)

```
<?php
$outputPath = "Desktop\\Exported-with-keywords";
$stagsiDataPath = "Database\\";
$db = new PDO("sqlite:$stagsiDataPath/Stagsi.sqlite");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$getTags = $db->prepare("
  SELECT Title
    FROM ObjectTags ot
    JOIN Tags t
      ON ot.TagRowId = t.RowId
   WHERE ot.ObjectRowId = ?
");

$stmt = $db->prepare("SELECT RowId FROM Tags WHERE Title =
'Author'");
$stmt->execute();
list($authorParentId) = $stmt->fetch();

$stmt = $db->prepare("SELECT RowId FROM Objects WHERE Format =
'jpg'");
$stmt->execute();

while ($row = $stmt->fetch()) {
  list($rowid) = $row;

  $getTags->bindValue(1, $rowid);
  $getTags->execute();

  $base = "$stagsiDataPath/".floor($rowid / 1000)."/$rowid";
  $file = is_file($base) ? file_get_contents($base) : "$base.jpg";
```

```
  $output = "$outputPath/$rowid.jpg";
  copy($file, $output);

  $iptc = '';

  while ($tag = $getTags->fetch()) {
    list($title, $parent) = $tag;
    // 0x02 is record ID, 0x50 and 0x19 are dataset IDs (a byline, a
keyword).
    $prefix = $authorParentId == $parent ? '1C0250' : '1C0219';
    $iptc .= pack('H*n', $prefix, strlen($title)).$title;
  }

  file_put_contents($output, iptcembed($iptc, $output));
}
```

EXIF specification: www.exif.org/Exif2-2.PDF.

IPTC specification with XMP node names: iptc.org/std/IIM/4.1/specifi….

Distributed (Shared) Database

There are several ways to allow multiple people (computers) to work with the same *Stagsi* database. However, in all cases only 1 user must be modifying the database at the same time, otherwise changes of other users will be lost.

- Windows network share. Very easy to use (built into Windows) but also easy to break. High-speed connection required (LAN or enterprise VPN).
- Web version. Very easy to access (using any desktop or mobile web browser) and only you (author) can modify the database.
- Version control system – primarily *git*. Requires some basic knowledge but is very reliable, distributed, doubles as a backup solution (can revert to any point in time and/or compare changes).

The Tools menu can help with the VCS approach – create a simple `.bat` file and place it within the *Tools* directory in the database folder to get a new menu item which will commit and push (publish) your changes in two clicks (if you don't want to enter the message every time, add `-m YOUR_MESSAGE` after `git commit`):

{} Tools/Git add, commit and push.bat • Skip this example (#exUV) • Browse on GitHub • Adjustments (1)

```
git add -A
git commit
git push
rem or:
git push -m YOUR_MESSAGE
```

Database Compartments

Generally, a single database wins over multiple small databases: it has a unified tag tree, ability to search among everything on one page, etc. However, extremely big databases (100,000s of objects) or portable databases (or BYOD systems) will often combine files in linked mode from multiple sources (e.g. USB HDDs or network shares), some of which may not be available all the time (e.g. only at home

or at work).

Stagsi doesn't mind missing files but two problems arise:

- If you have background integrity checks enabled (as they are by default and you are encouraged to leave them so), missing files may be marked `Corrupted` when they appear in your search results. This might be annoying.
- Files from unavailable sources appear in the *Object Browser* even though you know that such files are expected to be missing in your current environment and you'd rather have them hidden by default.

The `/pick` command may be used to address both issues:

- Come up with a list of "compartments" (possible environments) and a criteria to filter sources available in each of them. For example, if you have an MP3 collection at home then your "work PC" environment must exclude all MP3 objects, i.e. `-MP3` query. Or the other way around: if your work environment only has PDFs and DOCs then the query is `PDF | DOC`.
- Create a Windows shortcut for each compartment (environment): `Stagsi.exe /pick /force "YOUR ENVIRONMENT QUERY"`. For example: `work.lnk` launching `Stagsi.exe /pick /force "PDF | DOC"`.

As a result, *Stagsi* will ensure that all results no matter the user's search query match the `/force`'d one and consequentally mismatching (= unavailable) files will never be queued for integrity checks (which happens only when an object is displayed) and marked `Corrupted`.

Shared Per-User Database

Suppose you have a single large database and multiple users which need to work with it, each in their own context (search tabs, window positions, etc.). For example, a PC is used by several people and you want them to use the same underlying database files (like `Stagsi.sqlite`) while keeping their contexts (JSON files) per-user (in `%APPDATA%`).

Stagsi doesn't allow splitting the database natively but NTFS symlinks and directory junctions can help you. First, move the database into a shared folder, e.g. to `C:\Stagsi`. Ensure `Bootstrap.json` points to some per-user location (e.g. to `%APPDATA%\Stagsi` as it does by default in non-portable versions). Then, in every per-user folder create symlinks and junctions so that:

```
%APPDATA%\...\Stagsi.sqlite ===points to===> C:\Stagsi\Stagsi.sqlite
%APPDATA%\...\Settings.json ===points to===> C:\Stagsi\Settings.json
%APPDATA%\...\0 ===points to===> C:\Stagsi\0
%APPDATA%\...\1 ===points to===> C:\Stagsi\1
...
```

Files which remain not symlinked (such as `Windows.json`) will be created in the per-user location, not in the shared folder. When *Stagsi* opens such a database (not by the shared path – `C:\Stags\Stagsi.sqlite` – but by a per-user location) it will read linked files and folders (`Stagsi.sqlite`, etc.) from `C:\Stagsi` and read all other files from the per-user location (in `%APPDATA%`).

- You can disallow users to modify the database by revoking their write permission on the shared

folder using the *Security* tab of *Windows Explorer*'s folder *Properties*.

- Remember to create enough junctions to numbered folders in advance (`0`, `1`, etc.) and/or increase `FolderSize` if you are allowing users to import new files, else part of the database may end up in the per-user location and will break for other users.
- You can easily create symlinks using this *Windows Explorer* extension: schinagl.priv.at/nt/hardlinkshellext.... Note: it also allows creating hardlinks but *Stagsi* overwrites some data files so this kind of links will not work.

Attachments

Let's imagine you have a collection of video lessons or audio podcasts. You want to tag them, and you import them as usual. But while watching or listening to them, you might want to keep notes or create bookmarks at different positions (see our VLC extension), and you want them to remain connected to the original file.

One way is to import your notes into *Stagsi* and use tags to group notes with their video files. But this is tedious:

- You'll need a unique tag per every lesson.
- Notes are dynamic by nature, and when you change them – they are marked as "corrupted".
- You might not be interested in tagging nodes or otherwise seeing them in *Stagsi* but if imported, they will pollute search results.

Attachments offer a better way to do this. They are not part of the usual *Stagsi* operations – integrity checks, tagging, search, etc. They don't have thumbnails and you cannot "import" them. They are regular files and folders placed into special "compartments" (subfolders) that *Stagsi* reserves for every object in its database.

Attachments are managed via object's context menu in the browser. You are also free to manipulate each "compartment" folder via *Windows Explorer* or other program, even while *Stagsi* is running.

Attachments processes are started within CommonScriptEnv which is particularly useful for scripts and `.bat` files.

"Has Attachments" Tag

Stagsi doesn't track attachments and doesn't automatically maintain a special tag like "Custom Thumbnail". As a result, you can't search based on whether an object has attachments or not. This is because attachment directories may be (and are even encouraged to be) changed from the outside of *Stagsi*, e.g. using *Windows Explorer*.

If you need such a tag but do not want to assign it manually, use the following PHP script:

- First, create a tag that will serve as "Has Attachments" that you will use in search queries. Change the value of `$hasAttachmentsTagRowid` variable to this tag's `RowId`.
- Also change the value of `$stagsiDataPath` to the path of the directory with `Stagsi.sqlite`.
- Call the script while *Stagsi* is not running each time you need to update this tag (add it to objects with attachments and remove it from ones with none).

- If you run the script from the Tools menu, it will close *Stagsi* automatically.

Tools/Update Has Attachments tag.php • Skip this example (#exRN) • Browse on GitHub • Adjustments (2)

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

$hasAttachmentsTagRowid = 12;
$stagsiDataPath = "C:/foo/stagsi_debug/db";

$pid = escapeshellarg(getenv('PID'));
exec('taskkill /pid '.$pid);
do {
  exec('tasklist /fi "pid eq '.$pid.'" | find "INFO"', $o, $c);
  usleep(100000);
} while ($c);

$db = new PDO("sqlite:$stagsiDataPath/Stagsi.sqlite");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$db->exec("DELETE FROM ObjectTags WHERE TagRowId =
$hasAttachmentsTagRowid");

$stmt = $db->prepare("INSERT INTO ObjectTags (ObjectRowId, TagRowId)
VALUES (?, $hasAttachmentsTagRowid)");

foreach (scandir("$stagsiDataPath/Attachments") as $group) {
  if (ltrim($group, "0..9") !== "") { continue; }

  foreach (scandir("$stagsiDataPath/Attachments/$group") as $id) {
    if (ltrim($id, "0..9") !== "") { continue; }

    if (count( scandir("$stagsiDataPath/Attachments/$group/$id") ) >
2) {
      $stmt->bindValue(1, $id);
      $stmt->execute();
    }
  }
}
```

Attached Queries

Suppose you have a bunch of videos grouped by series. For example:

```
Black Mirror
  Season 1
    The National Anthem
    Fifteen Million Merits
    The Entire History of You
  Season 2
    Be Right Back
    White Bear
    The Waldo Moment
    White Christmas
```

You have many top-level tags (“Westworld”, “Doctor Who”, etc.) and each has numerous sub-tags and sub-sub-tags. You also have a special tag “Cover” which is assigned to exactly 1 item of the series

(for example, first season's first episode or the "best" episode of the series). When you're looking for something to watch, you include this "Cover" tag into the query so that the same series does not appear multiple times in the results.

When you've decided on a series, you may want to "enter" into it, searching for its episodes only. This involves:

- Removing the "Cover" tag from the search query.
- Adding a format tag, e.g. `MP4` (in case the database has not just videos but also audio, e.g. OST, from the same series with the same top-level tag).

If you do this by hand, it will quickly become tedious.

First, you can create an attached `.bat` file for every object which is part of some logical group (e.g. same video series). It would open one or more tabs in *Stagsi*. Here, first line opens a tab with *Black Mirror* videos while the second line opens its sound track:

```
"%STAGSI%" /search "Black Mirror" MP4
"%STAGSI%" /search "Black Mirror" MP3
```

Advantage: flexibility; you can put any command and have it called with a few keystrokes. Disadvantage: the same script should be attached to every object and if edited must be updated everywhere (but this can be rectified to a point with fsutil).

Second, you can use Tools if you know that all your video-related queries are constructed the same way (e.g. "Video" tag + name of the series under the "Series" root tag, for example: `Video Series/"Black Mirror"`). Name this script `Selection &episodes.php` and call it quickly with Alt+T,Alt+E:

Tools/Selection &episodes.php • Skip this example (#exQZ) • Browse on GitHub • Adjustments (2)

```
<?php
$selection =
json_decode(strstr(file_get_contents(getenv('selection')), '{'));
$seriesTagRowId = null;
$tags = [];

foreach ($selection->Tags as $tag) {
  $tags[$tag->RowId] = $tag;
  if (!$tag->ParentRowId and $tag->Title === 'Series') {
    $seriesTagRowId = $tag->RowId;
  }
}

$queries = [];

foreach ($selection->ObjectTags as $obj) {
  if ($tags[$obj->TagRowId]->ParentRowId === $seriesTagRowId) {
    $queries[$obj->TagRowId] =
      "\"{$tags[$seriesTagRowId]->Title}\"/\"{$tags[$obj->TagRowId]-
>Title}\"";
  }
}

$query = '('.join(' | ', $queries).') Video';
exec(escapeshellarg(getenv('stagsi')).' /search '.esc($query));
```

```
// escapeshellarg() simply drops "s.
function esc($s) {
  return '"'.preg_replace('/"/u', '""', $s).'"';
}
```

You can of course combine both approaches, using *Tools* for all but some objects requiring special treatment via an attached script.

Icon Variants

Let's assume you have a database of pretty icons for GUI design. Raster icons typically use the PNG format but you may have ICOs too, and even those PNGs can be of different sizes (say 16x16 and 32x32). It makes sense to import only one particular variant into *Stagsi* (say 32x32 PNGs) and add others as attachments to avoid polluting search results with icons different in dimensions only.

But what if you got thousands of icons? For example, the free *FatCow* iconset (an excellent one, by the way – www.fatcow.com/free-icons) has almost 4000 icons, in 11 color variations, that's not including ICOs. Attaching them all by hand would be insane.

Here's a helper script to do that for *FatCow*. It will work for other cases as long as the variations are placed into a single folder (e.g. `FatCow_Icons16x16` for 16x16 PNG, `FatCow_Icons16x16_Color\Blue` for same images but in a blue theme, etc.). Feel free to adapt it to your needs:

Database/Attachments.php • Skip this example (#exRB) • Browse on GitHub • Adjustments (5)

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

$attPath = 'C:/Stagsi DB/Attachments';

$db = new PDO("sqlite:$attPath/../Stagsi.sqlite");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$getStmt = $db->prepare("SELECT RowId FROM Objects WHERE Hash = ?");

$walk = function ($path, $basename) use ($getStmt, $db, $attPath) {
  foreach (scandir($path) as $icon) {
    if (is_file("$path/$icon")) {
      $orig = "F:/Icons/FatCow/FatCow_Icons32x32/$icon";

      $getStmt->bindValue(1, md5_file($orig));
      $getStmt->execute();
      $id = $getStmt->fetchObject()->RowId;
      $getStmt->closeCursor();

      $atts = "$attPath/".floor($id / 1000)."/$id";
      is_dir($atts) or mkdir($atts, 0755, true);
      copy("$path/$icon", "$atts/$basename.png");
    }
  }
};

// This script assumes that Stagsi database has 32x32 PNGs imported
and you
```

```
// want to add (attach) 16x16, gray 16x16 and gray 32x32 as
variants.

$walk("F:/Icons/FatCow/FatCow_Icons16x16", '16x16');
$walk("F:/Icons/FatCow/FatCow_Icons16x16_Grey", 'Disabled 16x16');
$walk("F:/Icons/FatCow/FatCow_Icons32x32_Grey", 'Disabled 32x32');
```

Web Database Viewer

Below is a PHP database browser. It's extremely simplistic but can serve as a basis for a full-scale implementation (but note that *Stagsi* already includes an easy sharing function with a similar online browser).

Pertinent features and cautions about the below code:

- Thumbnails are proxied through the script, to allow the database to reside outside of the web server's root.
- Only Latin symbols are properly supported in search queries. This is due to case folding performed and lack of Unicode "string-to-lower-case" function in *SQLite*.
- Search query is parsed using our public domain parser. This is the only dependency of this script (besides PDO with SQLite driver, of course) and it can even be removed.
- Object filtering (search query) is very basic and not optimal. It doesn't handle recursive terms (`~tag`), doesn't use the full title path (`parent/tag`), doesn't care for unique/duplicate titles (only uses last path title given by the user) and matches tags by titles, not IDs in SQL.
- Additionally, because it's just a single query with `JOIN`'s and a `WHERE`, when tag filter is in effect – returned list of tags will never have tags not mentioned in the query. All objects will have "tag list" of just one "foo" when searching for "foo" tag.

With this, the code:

Misc/Web viewer.php • Skip this example (#exGJ) • Browse on GitHub • Adjustments (3)

```
<?php
$stagsiDataPath = 'C:/Stagsi Demo/db';

function nodeToWHERE(Stags\Query\Node $node, PDO $db) {
  if ($node instanceof Stags\Query\ListNode) {
    $res = [];
    foreach ($node->children as $child) {
      $res[] = '('.nodeToWHERE($child, $db).')';
    }
    return join($node->isAnd ? ' AND ' : ' OR ', $res);
  } else if ($node instanceof Stags\Query\IdNode) {
    return 'o.RowId '.($node->isNegative ? '<>' : '=').' '.((int)
$node->id);
  } else if ($node instanceof Stags\Query\TagNode) {
    if ($node->isRecursive) {
      throw new Exception("Recursive (~) tag search is not
supported.");
    }
    return 'LOWER(t.Title) '.
           ($node->isNegative ? '<>' : '=').' '.
           strtolower($db->quote(end($node->path)));
  } else {
    throw new Exception("Unknown query node type.");
```

```
  }
}

set_error_handler(function ($severity, $msg, $file, $line) {
  throw new \ErrorException($msg, 0, $severity, $file, $line);
}, -1);

extract($_REQUEST + [
  'query'   => '',
  'sort'    => 'rowid',
  'desc'    => true,
  'view'    => 'thumbs',
  'thumb'   => '',
  'page'    => 1,
], EXTR_PREFIX_ALL, 'r');

if ($r_thumb = (int) $r_thumb) {
  chdir($stagsiDataPath);
  $path = floor($r_thumb / 1000)."/[cut]$r_thumb.*";
  $files = glob($path);
  $files = array_combine(array_map(function ($f) { return
basename($f)[0]; }, $files), $files);
  $files += ['c' => '', 'u' => '', 't' => ''];
  $file = $files['c'] ?: ($files['u'] ?: $files['t']);
  $info = getimagesize($file);
  header("Content-Type: $info[mime]");
  readfile($file);
  exit;
}

$db = new PDO("sqlite:$stagsiDataPath/db.sqlite");
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

// Obtain from https://github.com/Soletude/Stags-Query-Parser
require_once 'Stags-Query-Parser/Stags.Query.php';
$where = nodeToWHERE((new Stags\Query\Parser($r_query))->parse(),
$db) ?: '1';

ltrim($r_sort, 'a..z') === '' or $r_sort = 'rowid';
$sort = "o.$r_sort";
$desc = $r_desc ? 'DESC' : 'ASC';

$sql = <<<SQL
SELECT o.RowId, o.Title, GROUP_CONCAT(t.Title, '"') TagTitles
  FROM Object o
  JOIN dataobjecttags ot
    ON ot.dataobject_rowid = o.RowId
  JOIN Tags t
    ON t.RowId = ot.tag_rowid
 WHERE $where
 GROUP BY o.RowId
 ORDER BY $sort $desc
 LIMIT :start, :count
SQL;

$stmt = $db->prepare($sql);
$stmt->bindValue('start', ($r_page - 1) * 100);
$stmt->bindValue('count', 100);
$stmt->execute();

$qs = '?'.$_SERVER['QUERY_STRING'].'&';
?>

<!DOCTYPE html>
<html>
```

```
  <head>
    <title>Simple Stagsi Web Viewer</title>
    <style>
      .group { margin-right: 1em; }
      .results.thumbs img { height: 100px; }
      .results.list { text-align: center; }
    </style>
  </head>
  <body>
    <h1><?=htmlspecialchars($stagsiDataPath)?></h1>

    <form action="">
      <label class="group">
        <b>Search query:</b>
        <input value="<?=htmlspecialchars($r_query)?>" name="query">
      </label>

      <span class="group">
        <b>Sort by:</b>
        <select name="sort">
          <option value="rowid" <?=$r_sort == 'rowid' ? 'selected' :
''?>>Object ID</option>
          <option value="random" <?=$r_sort == 'random' ? 'selected'
: ''?>>Random</option>
          <option value="title" <?=$r_sort == 'title' ? 'selected' :
''?>>Title</option>
          <option value="filesize" <?=$r_sort == 'filesize' ?
'selected' : ''?>>File size</option>
          <option value="format" <?=$r_sort == 'format' ? 'selected'
: ''?>>Format</option>
        </select>

        <label>
          <input type="hidden" name="desc" value="0">
          <input type="checkbox" name="desc" value="1" <?=$r_desc ?
'checked' : ''?>>
          Descending
        </label>
      </span>

      <span class="group">
        <b>View as:</b>
        <label>
          <input type="radio" name="view" value="thumbs" <?=$r_view
=== 'thumbs' ? 'checked' : ''?>>
          Thumbnails
        </label>
        <label>
          <input type="radio" name="view" value="list" <?=$r_view
=== 'list' ? 'checked' : ''?>>
          List
        </label>
      </span>

      <button type="submit">Apply</button>
    </form>

    <p>
      <a href="<?=htmlspecialchars("$qs&page=".($r_page + 1))?
>">Next page</a>
      |
      <?php if ($r_page > 1) {?>
        <a href="<?=htmlspecialchars("$qs&page=".($r_page - 1))?
>">Previous page</a>
      <?php }?>
```

```
      |
      <a href="?">Reset all</a>
    </p>

    <div class="results <?=htmlspecialchars($r_view)?>">
      <?php while ($row = $stmt->fetchObject()) {?>
        <?php if ($r_view === 'list') {?>
          <hr>
          <a href="<?=htmlspecialchars("?thumb=$row->RowId")?>">
            <img src="<?=htmlspecialchars("?thumb=$row->RowId")?>">
          </a>
          <p><?=htmlspecialchars($row->Title)?></p>
          <p>
            <?php foreach (explode('"', $row->TagTitles) as $tag) {?
>
              [ <a href="<?=htmlspecialchars("$qs&query=\"$tag\"")?
>">
                <?=htmlspecialchars($tag)?></a> ]
            <?php }?>
          </p>
        <?php } else {?>
          <a href="<?=htmlspecialchars("?thumb=$row->RowId")?>">
            <img src="<?=htmlspecialchars("?thumb=$row->RowId")?>"
                 title="<?=htmlspecialchars($row->Title)?>">
          </a>
        <?php }?>
      <?php }?>
    </div>
  </body>
</html>
```

Performance Tips

One of the main (and only) challenges in *Stagsi* is thumbnail loading. If you are browsing through a list of 1000 objects and every object's thumbnail is 100 KiB (a modest estimate) – then *Stagsi* needs to load about 1000×100 KiB = 100 MiB of data really fast – and then to render it.

For traditional HDDs (spinning hard-disk drives) 100-150 MiB per second is the total limit they may provide in ideal conditions (only 1 process is using the drive, the file system is not fragmented, all files are near each other "physically").

Modern SSDs and especially NVMe's are inexplicably better at the kind of tasks *Stagsi* needs but may still fall short in some circumstances (large thumbnail files, multiple processes accessing the disk simultaneously, real-time antivirus filters, low-end hardware, etc.).

Therefore it's recommended to all users to consider the following factors contributing to smooth *Stagsi* browsing, in order of importance:

- Free RAM and the `CacheImagesInMemory` and `MaxImageCacheSize` settings. *Stagsi* loads thumbnails into memory from disk in advance, caching all of them for small databases or regenerating the cache each time you switch tabs for large ones. The more Gigabytes you can give to *Stagsi*, the better. This is the biggest performance factor.
- Be always-on: if you're using *Stagsi* often, do not exit it but minimize to tray (this is enabled by default) so that its cache is not invalidated by a "cold" start. The same approach is used by *Windows Explorer* (obviously), *Microsoft Office* and *Google Chrome*.
- Good hardware, in particular – disk, especially for databases that cannot fit into RAM (either

because they're very big or because the amount of RAM is small). Even though *Stagsi* has been successfully used over USB, good hardware will make the experience significantly better.

As a somewhat reverse advice, if you have a very fast disk (NVMe) and a large database (a lot of large images that cannot fit in memory at once) or you have to constantly close *Stagsi* process then you may actually benefit from disabling thumbnail cache (`CacheImagesInMemory`) to remove unnecessary burden from your disk and CPU while still maintaining an adequate performance.

- When changing multiple tags at once of a large selection (thousands of objects) use the modal *Tag Selection* (`Space`) dialog rather than *Object Tags* (`Ctrl+G`). This is because the modal dialog applies changes to all tags at once in one operation, not one operation per each tag as via *Object Tags*.
- Optimize the thumbnails with `MaxThumbWidth`, `ThumbFormat` and `ThumbQuality` settings. If you don't zoom out too much, lower `MaxThumbWidth`. If you want best quality, don't set `ThumbQuality` to 100 but see if 90 works as good, or consider switching to PNG format. All of this will result in smaller thumbnail file sizes, faster loading and better performance (as explained above).
- Manually tailor the thumbnails to your use. Standard settings are limited and suitable for configuration-free, general-purpose databases. However, if you have many pixel art images, you can use a tool like *XnView* to generate indexed PNG images with short palettes, dramatically reducing their file size, speeding up the performance without compromising image quality.

Portable Builds

Stagsi doesn't have to be installed on the target system. Moreover, it can be configured to avoid writing to any locations outside of its directory.

Note: by default *Stagsi* is coming with a tiny automatic update program which does write to the registry and keeps logs and partially downloaded updates in the current user's directory (`%APPDATA%`). If you want to avoid that, rename `Sup.exe` (so you can update *Stagsi* manually by running the renamed file) or delete it (and update by extracting new portable version's archive from the website).

Aspect	Explanation	Portable considerations
Registry	If the user has accepted the license agreement on the start-up then the `EulaAccepted` key is created under `HKCU\Software\Soletude\Stagsi`.	Due to legal requirements, this key cannot be disabled.
Logs	When *Stagsi* is launched, it starts writing logs even before database settings are read. The logs go to *Stagsi*'s folder `Logs` subfolder, or to `%APPDATA%\Soletude\Stagsi\Logs` if the former is not writeable (as is the case when it's installed in `Program Files`).	Ensure *Stagsi* can write to its `Logs` subfolder.
Database	The location of the database is specified in `DataPath` key of `Bootstrap.json`'s in the *Stagsi*'s folder. By default and when it's missing or bad, `%APPDATA%` is used. The value is relative to *Stagsi*'s folder.	Set `DataPath` key of `Bootstrap.json` to `Database` (a subfolder of *Stagsi*).

Write access	On launch, *Stagsi* tests if the database file is writeable and automatically enables `/ro` (read-only) mode if it isn't.	*Stagsi* won't break if placed in a read-only location.

Using an External (USB) Drive

Stagsi cares little for where its data files are located. For portability purposes, the database can be placed on an external USB drive, possibly even with *Stagsi* binaries to avoid installing the program on the target system (see the *Portable Builds* section for details).

Common questions about using an external drive for *Stagsi*:

- `Drive lifespan/wearing.` *Stagsi* does not produce any overly heavy activity to significantly affect the drive's lifespan, be it an HDD or SSD. It does read a lot of data (depending on your database's size, perhaps around the same as watching a few high-quality movies) but drive wearing is normally measured in the amount of data written, and *Stagsi* only writes it intensively during import, which is about the same as copying all imported files onto the hard drive in *Windows Explorer*.
- `Stagsi becoming unresponsive after an idle period.` For better power management and damage protection, HDDs may "spin down" when they have not been used for a while (normally – about 10 minutes). This happens for internal HDDs as well. If after leaving *Stagsi*'s window open for a while you notice it's not responding for several seconds when switching back to it – it's most likely due to the drive "warming up". You might even hear a characteristic sound coming out from your HDD enclosure. This is perfectly normal but might be annoying. Most manufacturers provide a tool for changing the idle period or disabling it completely – for example, *WD Drive Utilities* by *Western Digital*.
- `Sluggish scrolling, etc.` *Stagsi* needs as fast disk access as possible. External HDDs often have very poor performance, which may or may not your usage (depending on the database size, thumbnail size, amount of RAM for file caching, etc.). To improve the situation, consider an external SSDs with USB 3 support, or an internal drive (even if HDD). See also *Performance Tips*.

Version Numbers

Internally, *Stagsi* is an interface (hence the "-i") to *Stags* ("Soletude's Tagging System") which is internally called "the library" (and *Stagsi* called "the host program"). Both parts have separate versions exposed in Windows file properties (`Soletude.Stags.Library.dll` and `Stagsi.exe`) and in the database's `Stags` table (shown in the *Database* panel in the UI).

`LibraryVersion` indicates the version of the database schema and is used in the migration process. It consists of two numbers: *major* and *minor*. DLL properties also include *build number* but it is ignored for the purpose of determining if two schema versions are the same. This version does not appear outside of the database field and DLL properties.

`HostVersion` indicates the program's "overall" version as *major.minor.build number*, used in different places, e.g. in the *Copy* command's JSON. Given two host versions, if their *build numbers* are different but other components are the same then it's essentially the "same" *Stagsi* version, else the version having a larger *build number* is more recent (thus nothing needs to be calculate based on *major* and *minor* numbers, they only need to be compared as is). `HostVersion` in the database is purely informational.

Loosely following semver.org, *major* is incremented whenever a significant change occurs, possibly breaking old scripts, plugins, etc.; *minor* is incremented routinely for each new release; *build number* is used for internal accounting purposes.

Alternative Software

Stagsi is not the only solution in the field of data management but we believe the competition has critical flaws and so we're not afraid of talking about it. If you're not happy with *Stagsi* – get in touch with us go.soletude.ca/stagsi/forum. If you know another alternative – let us know and we will evaluate it.

- *Adobe Bridge* – the industry standard among artists. It's huge (installer is around 1 GiB), can only manage some images (PNG and JPEG but not GIF) and videos, alters file data to store tag information (consequently, has no separate database and cannot detect duplicates), offers no search query language. Advantages: cross-platform, supports videos and is free. With all the limitations, *Bridge* was the number one prototype we had in mind when designing *Stagsi*.
- *Apple iTunes* – another industry standard, this time among audiophiles. It's reasonably large (installer is around 250 MiB), heavily wired into the Apple's ecosystem, only supports audio files, alters file data. Advantages: great usability (even though it looks alienate on Windows), cross-platform, free.
- *Tabbles* tabbles.net – a worthy product similar to *Stagsi* in many aspects but costs ~$20-$79 per year (as of October 2019), heavier (over 100 MiB if you include the dependencies – a local SQL Express Server), with a very limited free version (comparing to *Stagsi*). Advantages: an old project, online collaboration, supports any kind of files, separate meta-data storage (at last!), automatic tagging (but less powerful than *Stagsi*'s wildcard, regexp and JSON-based rules), integration with *Windows Explorer* and web browsers, somewhat hackable.
- *Tag Spaces* www.tagspaces.org – an open-source commercial product (from $44 per year of updates as of October 2019). Stores tag information in file names (!), hardly works with more than a dozen of tags (no tag tree, hotkeys, sluggish UI), tries to mimic a file browser, extensions require an Enterprise license. Advantages: cross-platform, open-source, integration with web browsers, portable.
- *Windows Live Photo Gallery* – was part of Windows since Vista but was dropped in 2017. Advantages: blended nicely with *Windows Explorer*, was fast and easy to use, had batch tagging features and some useful photo tools.

Database From Scratch

A database can be entirely auto-generated, without relying on *Stagsi* GUI or executables at all. This has many uses – one is data "scraping", i.e. taking of data from an arbitrary resource in an arbitrary format and presenting it uniformly using *Stagsi*.

Freelancer.com

Most websites present their information differently even though the underlying data may have the same structure and then fail at providing adequate means to work with it.

Freelancer.com is a resource where you can create design contests (among other things); you get a listing of submitted works – a plain simple grid of images with few sorting options, no filtering, fixed

page size, fixed thumbnail dimensions, practically no rating capabilities. A contest may see thousands of submissions and reviewing them (especially if you have other team members) becomes a painful experience.

This can be solved by importing results into *Stagsi* – gaining sharing (*git*), tagging and filtering, sorting, backing up and so on. First, open your contest's page, sort it in some stable manner (e.g. by time submitted), open the Console (`Ctrl+Shift+I`), paste this script and run it (`Enter`):

Import/Freelancer.com grabber.js • Skip this example (#exSJ) • Browse on GitHub

```
var f = function () {
  document.querySelectorAll('#entry-cards-list figure
img.ContestCard-image')
    .forEach(img => a.push(img.getAttribute('src'))
  var pages = document.querySelectorAll('.btn.Pagination-link')
  pages[pages.length - 1].click()
  var t = setTimeout(f, 1500)
  window.onunload = () => clearTimeout(t)
}
var a = []; f()
```

Once it walks to the last page, run this in the Console: `a.join('\n')` – it will output a list of thumbnail URLs. Copy/paste them into a text file and feed that file to any download manager, such as *GNU Wget*:

```
wget -iurls.txt
```

Finally, import the downloaded images into *Stagsi* as usual.

It's safe to repeat this process multiple times because thumbnails do not change and so already imported images are detected as duplicates (same data hash).

Unicode Table

Unicode is a standard for encoding human texts. It strives to encompass all writing system in use today, in the past and in the future (emoji) and so the number of symbols known to this standard is huge (tens of thousands). Each symbol is assigned a set of properties, such as "Latin plane" or "Right-to-left script".

Below is a PHP script filling an empty database (copied from `Skeleton` folder) with *Unicode* symbols, creating thumbnails for each of them and assigning tags according to the symbol's properties. Things to note:

- Each object is a file with `.unicode` with type association set up such that opening it copies it to the clipboard.
- Many objects also have `.unicode` attachments – their variants, such as lower-case version ("q") of an upper-case symbol ("Q").
- The `Objects` table gains a custom column `Codepoint` (decimal number such as `81` for U+0051 – "Q") which is displayed in *Stagsi*'s *Properties* panel and can be copied with a single click.
- The `Stags` table gains a key `Unicode` holding the version of the standard on which the data is

based. It's displayed in the *Database* panel.

Database/Unicode.php • Skip this example (#exYD) • Browse on GitHub • Adjustments (3)

```
<?php
set_error_handler(function ($severity, $msg, $file, $line) {
  throw new ErrorException($msg, 0, $severity, $file, $line);
}, -1);

if (!class_exists('IntlChar')) {
  die('php_intl module is required.');
}

copy('C:/Program Files/Stagsi/Skeleton/Stagsi.sqlite',
'Stagsi.sqlite');
copy('C:/Program Files/Stagsi/Skeleton/Settings.json',
'Settings.json');
$settings = json_decode(file_get_contents('Settings.json'));
$settings->BitmapScalingMode = 'Fant';
$settings->ThumbFormat = 'png';
$perDir = $settings->FolderSize;
file_put_contents('Settings.json', json_encode($settings,
JSON_PRETTY_PRINT));

$bom = "\xEF\xBB\xBF";
$rootTagID = 1;

$assoc = [
  [
    "CallMode" => 1,
    "ExecutableList" => [
      "nircmd clipboard readfile \"%1\"",
    ],
    "Extension" => ".unicode"
  ],
];

file_put_contents('Associations.json', json_encode($assoc,
JSON_PRETTY_PRINT));

$db = new PDO('sqlite:Stagsi.sqlite');
$db->setAttribute(PDO::ATTR_ERRMODE, PDO::ERRMODE_EXCEPTION);

$selectTag = $db->prepare('SELECT RowId, * FROM Tags WHERE Title =
:title AND ParentRowId = :parent');
$selectRootTag = $db->prepare("SELECT RowId, * FROM Tags WHERE Title
= :title AND ParentRowId = $rootTagID");
$insertTag = $db->prepare("INSERT INTO Tags (Title, CreationTime,
\"Order\", ParentRowId, IconPosition) VALUES (:title,
STRFTIME('%s','NOW'), 0, :parent, 0)");
$insertObject = $db->prepare("INSERT INTO Objects (CreationTime,
Random, Hash, Title, Format, ThumbCropX, ThumbCropY, ThumbCropWidth,
ThumbCropHeight, FileSize, Codepoint) VALUES (STRFTIME('%s','NOW'),
RANDOM() % 2147483647, :hash, :title, :format, 0, 0, 0, 0, :size,
:codepoint)");
$insertObjectTag = $db->prepare('INSERT INTO ObjectTags
(DataObject_RowId, Tag_RowId) VALUES (:object, :tag)');

$ver = join('.', IntlChar::getUnicodeVersion());
$db->exec("INSERT OR REPLACE INTO Stags SET Key = 'Unicode', Value =
'$ver'");

try {
  $db->exec("ALTER TABLE Objects ADD COLUMN Codepoint INT NOT NULL
```

```
DEFAULT 0");
} catch (PDOException $e) {
  // Already added, ignore.
}

list($formatTagID) = createTag(['Unicode File']);
$db->exec("UPDATE Tags SET System = '_unicode' WHERE RowId =
$formatTagID");

$codepoints = [];

// Generation of 0x0000..0xFFFF (55346 codepoints) takes 30 minutes.
IntlChar::enumCharNames(0x0000, 0x0FFF, function ($codepoint) use
(&$codepoints) {
  $codepoints[] = $codepoint;
});

$count = 0;

foreach ($codepoints as $codepoint) {
  if (++$count % 500 == 0) {
    printf('%s / %s (%.1f%%)...%s', $count, count($codepoints),
      $count / count($codepoints) * 100,
      PHP_EOL);
  }

  $text = IntlChar::chr($codepoint);

  $insertObject->bindValue(':hash', md5($bom.$text));
  $title = IntlChar::charName($codepoint,
IntlChar::UNICODE_CHAR_NAME);
  $insertObject->bindValue(':title', $title);
  $insertObject->bindValue(':format', 'unicode');
  $insertObject->bindValue(':size', $bom.$text);
  $insertObject->bindValue(':codepoint', $codepoint);
  $insertObject->execute();
  $objectID = $db->lastInsertId();
  $objectGroup = floor($objectID / $perDir);

  linkTagsTo($objectID, $formatTagID);

  is_dir($objectGroup) or mkdir($objectGroup);
  file_put_contents("$objectGroup/$objectID.unicode", $bom.$text);
  drawText("$objectGroup/u$objectID.png", $text);

  $age = IntlChar::charAge($codepoint);
  $age[count($age) - 1] = 'Unicode v'.join('.', $age);
  createTagFor($objectID, array_merge(['Age'], $age));

  $digit = IntlChar::charDigitValue($codepoint);
  $digit < 0 or createTagFor($objectID, ['Digit', $digit]);

  $num = IntlChar::getNumericValue($codepoint);
  $num === ((float) -123456789) or createTagFor($objectID,
['Numeric', $num]);

  $dir = IntlChar::charDirection($codepoint);
  createTagFor($objectID, ['Direction', prettyConst($dir,
'CHAR_DIRECTION_')]);

  $names = [
    'Name' => IntlChar::UNICODE_CHAR_NAME,
    'Name/Alias' => IntlChar::CHAR_NAME_ALIAS,
    'Name/Choice Count' => IntlChar::CHAR_NAME_CHOICE_COUNT,
  ];
```

```
  foreach ($names as $tag => $const) {
    $name = IntlChar::charName($codepoint, $const);
    if ($name) {
      createTagFor($objectID, array_merge(explode('/', $tag),
[$name]));
    }
  }

  $name = IntlChar::charName($codepoint,
IntlChar::EXTENDED_CHAR_NAME);
  if ($name !== IntlChar::charName($codepoint)) {
    createTagFor($objectID, ['Name', 'Extended', $name]);
  }

  $cat = IntlChar::charType($codepoint);
  createTagFor($objectID, ['Category', prettyConst($cat,
'CHAR_CATEGORY_')]);

  $pair = IntlChar::getBidiPairedBracket($codepoint);
  $pair === $codepoint and $pair = false;
  $lower = IntlChar::tolower($codepoint);
  $lower === $codepoint and $lower = false;
  $upper = IntlChar::toupper($codepoint);
  $upper === $codepoint and $upper = false;
  if ($pair !== false or $lower !== false or $upper !== false) {
    $apath = "Attachments/$objectID";
    is_dir($apath) or mkdir($apath, 0750, true);
    $pair === false or file_put_contents("$apath/Pair.unicode",
$bom.IntlChar::chr($pair));
    $lower === false or file_put_contents("$apath/Lower.unicode",
$bom.IntlChar::chr($lower));
    $upper === false or file_put_contents("$apath/Upper.unicode",
$bom.IntlChar::chr($upper));
  }

  $block = IntlChar::getBlockCode($codepoint);
  // E.g. block ID 279 has no constant.
  createTagFor($objectID, ['Block', prettyConst($block,
'BLOCK_CODE_') ?: "#$block"]);

  $class = IntlChar::getCombiningClass($codepoint);
  $class and createTagFor($objectID, ['Combining Class', $class]);

  $closure = IntlChar::getFC_NFKC_Closure($codepoint);
  strlen($closure) and createTagFor($objectID, ['FC NFKC Closure',
$closure]);
}

echo "Wrote $count codepoint objects.", PHP_EOL;

function createTagFor($objectID, array $path) {
  $ids = createTag($path);
  linkTagsTo($objectID, end($ids));
}

function createTag(array $path) {
  global $rootTagID, $db, $selectTag, $selectRootTag, $insertTag;
  $ids = [];

  foreach ($path as $i => $title) {
    if (!$i) {
      $selectRootTag->bindValue(':title', $title);
      $selectRootTag->execute();
      $row = $selectRootTag->fetch();
```

```
    } else {
      $selectTag->bindValue(':title', $title);
      $selectTag->bindValue(':parent', end($ids));
      $selectTag->execute();
      $row = $selectTag->fetch();
    }
    if ($row) {
      $ids[] = $row['RowId'];
    } else {
      $insertTag->bindValue(':title', $title);
      $insertTag->bindValue(':parent', end($ids) ?: $rootTagID);
      $insertTag->execute();
      $ids[] = $db->lastInsertId();
    }
  }

  return $ids;
}

function linkTagsTo($objectID, $tagIDs) {
  global $insertObjectTag;

  foreach ((array) $tagIDs as $tagID) {
    $insertObjectTag->bindValue(':object', $objectID);
    $insertObjectTag->bindValue(':tag', $tagID);
    $insertObjectTag->execute();
  }
}

function prettyConst($value, $prefix) {
  static $consts;
  if (!$consts) {
    $consts = (new ReflectionClass(IntlChar::class))-
>getConstants();
  }
  foreach ($consts as $cname => $cvalue) {
    if (!strncmp($cname, $prefix, strlen($prefix)) and $cvalue ===
$value) {
      return ucwords(strtolower(strtr(substr($cname,
strlen($prefix)), '_', ' ')));
    }
  }
}

function drawText($file, $text) {
  $im = imagecreatetruecolor($w = 128, $h = 128);
  imagesavealpha($im, true);
  // Windows Explorer would show 100% transparent pixels as fully
opaque.
  // E.g. if 127 is changed to 126, the image would appear truly
transparent.
  // But it's an Explorer's quirk - Photoshop does show proper 100%
transparency.
  imagefill($im, 0, 0, imagecolorallocatealpha($im, 255, 255, 255,
127));
  $font = 'c:/windows/fonts/ARIALUNI.TTF';
  $size = 48;
  $box = imagettfbbox($size, 0, $font, $text);
  // 64x64
  //   1:7  ---  32:7
  //    |   '@'   |
  //   1:37 ---  32:37
  $x = round(($w - $box[2] + $box[0]) / 2);
  $y = $h - round(($h - $box[1] + $box[7]) / 2);
//if ($text==='@') {
```

```
//  foreach ($box as $i => &$ref) { $i % 2 and $ref += 30; }
//var_dump($text,$box,$x,$y);
//imagerectangle($im, $box[6], $box[7], $box[2], $box[3],
imagecolorallocate($im, 255, 0, 0));
//}
  imagettftext($im, $size, 0, $x, $y, imagecolorallocate($im, 0, 0,
0), $font, $text);
  imagepng($im, $file);
}
```