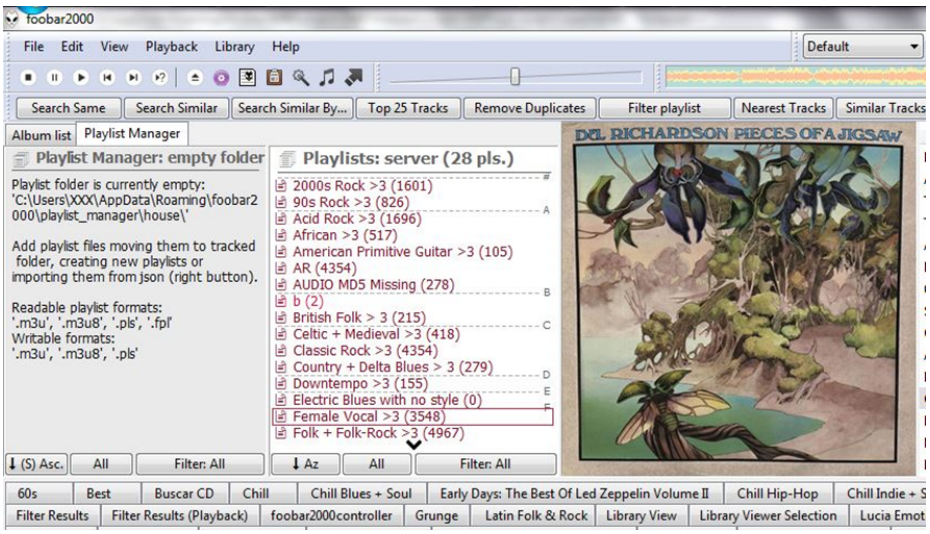
Playlist Manager

A PLAYLIST MANAGER FOR FOOBAR2000 USING SPIDER MONKEY PANEL AND JAVASCRIPT

BY

REGORXXX

AUTHOR OF "PLAYLIST-TOOLS-SMP", "WORLD-MAP-SMP", ETC.



Playlist-Manager-SMP
Published on GitHub
June 9, 2022

Contact me at: regorxxx@protonmail.com

Contents

Part I

Introduction

1 Why do we need a playlist manager?

1.1 Playlist tabs and UI limits

Foobar2000 already excels at library management, specially with plugins, but there is a hole in its set of features: playlist management. Playlists, by default, are entities always loaded within the UI, specially for those using playlist tabs. At some point, if there is a high number of them, the UI becomes cluttered with so many tabs that it becomes useless.

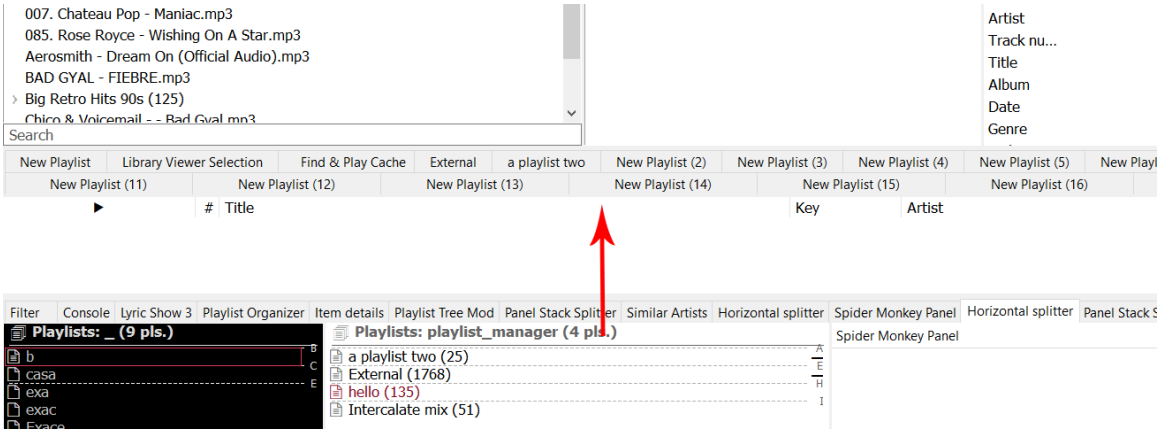


Figure 1: Playlist tabs become useless for a high number of playlists.

For this reason there has been some attempts of playlist manager UI plugins for DUI and CUI[1] which try to work around that with lists or playlist drop-downs. The problem is that those solutions are only workarounds... the playlists are still loaded within Foobar2000, so you must either completely remove the playlist tabs or clutter the UI with them. The managers make it easier to look for specific playlists, but the tabs dilemma remains the same.

And lets not talk about having thousands of playlists, then the only layout possible is a single tab for the active playlist and using a playlist list to switch between them. The problem with that layout is obvious. Are you able to switch between 2 or 3 playlist easily? No, that's what tabs are meant for, but since they are unusable in this case...

The problem seems to be that playlists are always loaded within Foobar2000. Why do you need a list of playlists if they are already loaded? Would not make it more sense if you were able to load a playlist -from the list- in the tabs on demand the same than you add a track to a playlist -from the library-?

[1] Default UI and Columns UI.

1.2 Auto-playlists slowdowns

Other of the problems experienced by advanced users appears during the prominent use of Auto-playlists. They are so great that it's easy that you end creating a lot of them... but then Foobar2000 magically becomes really slow at startup. Why? Because Auto-playlists query the library for updates, so every Auto-playlist instance equals to a query that is constantly checked on real time[2]... At startup is even worse, since all those great Auto-playlists have to be created at that point, thus requiring a lot of time until all are done.

Some examples to easily break your Foobar2000 experience if you add multiple instances of them :), enjoy:

> %rating% MISSING OR %last_modified% DURING LAST 1000 SECONDS
>
> %last_played% DURING LAST 19 MINUTES
>
> (%replaygain_track_gain% MISSING) OR (%style% MISSING) OR (%ALBUM DYNAMIC RANGE% MISSING)
>
> NOT ((%path% HAS "\-\") OR (NOT %path% HAS "{") OR (%comment% MISSING) OR (%title% IS -))

The problem -again- seems to be the design choice of playlists always being loaded within the program. If Auto-playlists were only loaded on demand, there would be no need to check them all at startup... neither constantly on real time. For this particular problem, there is a Spider Monkey Panel script which allows you to load Auto-playlists on demand: marc2003's Auto-playlist Manager.

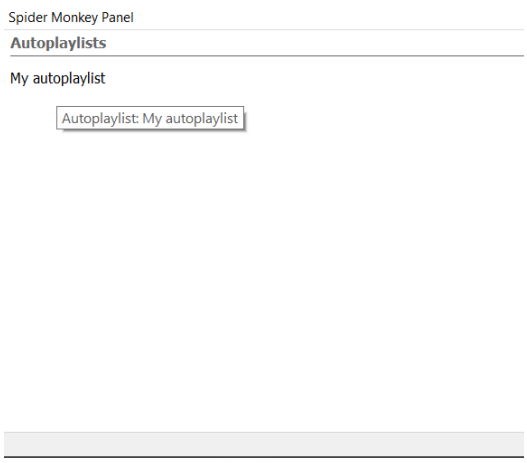


Figure 2: marc2003's Auto-playlist Manager.

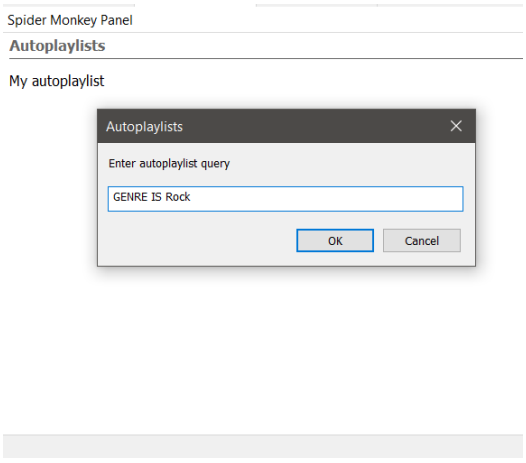


Figure 3: Editing a query.

[2]For ex. Check this thread: High Processor usage at Foobar Idle, just a single query which uses time ranges lower than 20 minutes are a problem.

1.3 Playlists are a bit auto-destructive

One of the consequences of playlist being an UI element by design, is that they only exist while being loaded within the UI. That seems a redundant pleonasm (pun intended), but is not: playlists are permanently deleted as soon as you close their instance in the UI. And that's a big thing[3]... Furthermore, while they may be 'restored' as long as you deleted the playlist on the same session, as soon as you close Foobar2000 the playlist history is gone. So it can not be undone.

The obvious problem is playlist not having a physical counterpart file managed in a non-destructive way. Well, to honor the truth, standard playlists do have a physical file somewhere in the profile folder... but:

1. They are in a **closed source format**.
2. They use an **non-human readable UUIDs as name**, so there is no way to know to which playlist they are linked to (aka good luck making backups or exporting an specific playlist).
3. The **files are only updated when shutting down the program**.
   (a) Changes between sessions are lost if there is a crash.
   (b) Playlists files are permanently deleted if you close a playlist.
   (c) New playlists are only saved when closing the program.
4. All the file management is done without indication, user intervention and in a non-transparent way.

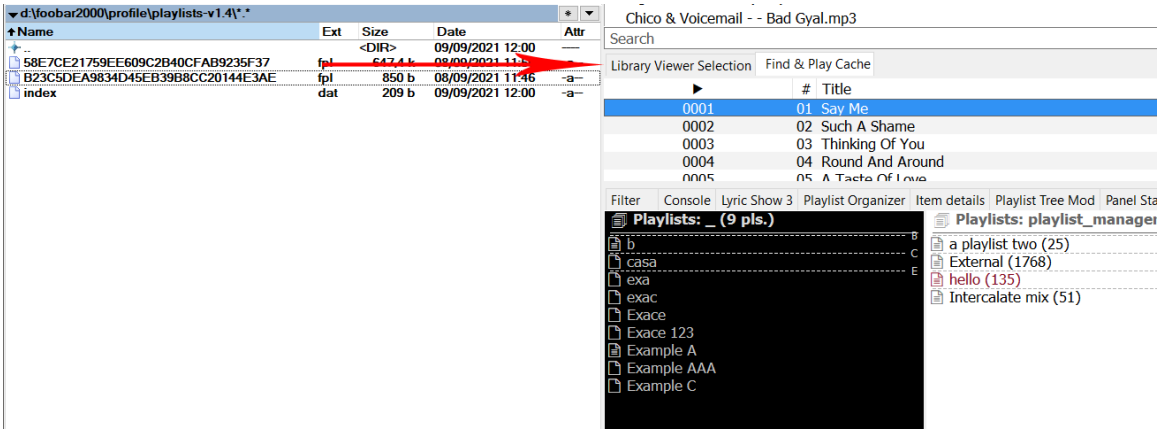


Figure 4: Uhm... yeah. Clearly '58E7CE21759EE609C2B40CFAB9235F37' belongs to the library viewer. Or is the other?

An ad-hoc solution for some of these problems is the use of another plugin to create automatic backups (including the physical playlists files and the database for unsaved changes): Autosave & Autobackup (foo_jesus).

[3]Do you imagine loosing your music files because you deleted the track from a playlist? Makes no sense, but translated to playlist files, that's they way they work within Foobar2000.

2 Multiple problems, one solution?

It seems there are partial solutions for Auto-playlists and others for standard playlists, but none for both... and in any case, either with native Foobar2000 or using plugins, some features are missing: adding labels to playlists, sub-folders to categorize them (instead of a plain list), searching, loading on demand (for standard playlists), easy exporting or syncing between the native foobar format and a plain text file[4]

Looks like you have to mix and match tons of plugins and scripts to manage different types of playlists, without an unified UI or manager and with basic features missing. Foobar is meant to manage libraries and tracks, not playlists[5].

So multiple problems and limitations have been discussed and now comes the point where a solution is proposed... and there is the place where this manager fits: a playlist manager which aims to work with playlist and Auto-playlists (without distinction) on the same panel, which only loads things on demand when required. Add some neat features to the mix like labeling, auto-tagging, auto-saving, syncing physical playlist files with loaded ones, backups, advanced exporting tools and integrity checks... and that's only a fraction of what can be done with it. Btw, marc2003's Auto-playlists are fully compatible and can be imported, in case you wonder.

2.1 Which playlists are compatible with it?

Short answer: All relevant.

Within the Playlist Manager context, playlists are virtual items (loaded in the UI) also linked to a physical file in the preferred format (.m3u8, .m3u, .pls, .strm or .fpl).

Auto-playlists, due to its nature (its content change according to the library they reside in), are saved into json format [VIII] in a file named accordingly to the folder tracked by the Playlist Manager[6].

.fpl playlists, similarly to Auto-playlists, are read only files... due to its closed source nature they are locked for further editing by default and metadata[7] is saved into the json file described previously.

In resume, writable formats are those that may be freely edited (.m3u8, .m3u, .pls) and readable-only formats are those that may be tracked by the manager with at least a minimum set of features[8] and the capability to load their tracks within foobar (.strm, .fpl or .json).

[4] Obviously nothing stops you to save your playlist manually using 'File\Save playlist...'. But how do you do that on 100 playlists?

[5] Not saying there is a player out there which does it better, so lets not even talk about having all those features!

[6] For ex. if the panel is set to track 'H:\My Music\Playlists', then the playlist json file (at foobar profile folder) will be at '.\js_data\playlistManager_Playlists.json'.

[7] Category, tags, lock status, etc.

[8] Metadata editing and conversion to a writable format.

Part II

Installation

3 Scripts files and dependencies

Spider Monkey Panel 1.5.2 or higher is required. Only stable releases are supported.

Copy all files from the zip into 'YOUR_FOOBAR_PROFILE_PATH\scripts\SMP\xxx-scripts'. **If the folders don't exist, create them. Any other path WILL NOT work without editing the scripts**. (see '_TIPS and INSTALLATION_vX.X.jpg'). Multiple scripts may share some files (specially helpers) so overwrite if asked to do so. Then load the script named **'playlist_manager.js'** into a SMP panel within foobar [5].

- For **standard installations**:
  'C:\Users\yourUser\AppData\Roaming\foobar2000\scripts\SMP\xxx-scripts\...'
- For **portable installations >= 1.6**: '.\foobar2000\profile\scripts\SMP\xxx-scripts\...'
- For **portable installations <= 1.5**: '.\foobar2000\scripts\SMP\xxx-scripts:.'

If you upgraded to >1.6 from an older portable version then it may be possible that the 'profile' folder does not exist. In such case you have to create it and move all the configuration folders/files to it, where they should reside (instead of the root of foobar2000 installation path). If you don't move all the configuration folders/files, then, on startup, default values will be used for things not found, probably "losing" the theme or other customization. You may "fix" it later moving the missing files which still reside in the root. May take some tries to do them all.

- **Some native folders and files which must be moved include:** index-data, js_data, component-updates, configuration, crash reports, user-components, foo_spider_monkey_panel, library, playlists, theme.fth, LargeFieldsConfig.txt, version.txt
- **Some extra folders from other components which must be moved include (non extensive list):** autobackup, dvda_metabase, foo_httpcontrol_data, foo_youtube, images, lastfm, python, sacd_metabase, vst-presets, yttm, minibar.db, playlist-tree-0.pts, playlist-tree-1.pts

Additionally, some fonts are required on the system. They can installed by double clicking on the '.ttf' files and then selecting the 'install' option (requires Admin rights):

- **Font Awesome** (v4.7.0 or greater): found at '._resources\fontawesome-webfont.ttf'

4 Portable installations tip

When the script finds it's being loaded within a portable installation, it will set the default paths using relative paths. It will also warn with popups and/or the console about the -non recommended- use of absolute paths on portable installations. For more info see [41].

5 Installation within Foobar2000's UI

Once all files have been installed to the right paths, along their dependencies, an Spider Monkey Panel must be added to the current layout anywhere on the UI. There are some minor differences between the Default UI (DUI) and Columns UI (CUI), but in both cases the layout can be edited using the menu **'View\Layout\Live editing'**:

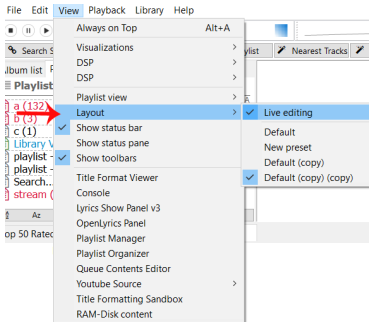


Figure 5: Editing the layout (CUI).

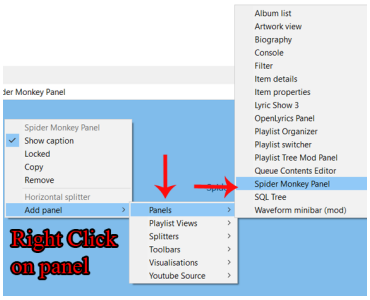


Figure 6: Adding a new SMP panel.

An SMP panel is only a blank UI panel with no further functionality until a '.js' script file is loaded. Multiple SMP panels can be added to a layout and they may point to the same script files without any problem; every panel is considered an independent entity. Multiple Playlist Manager panels may be added to the UI this way [40], either to have different views of the same playlists or different folders tracked. After the panel has been added to the current UI, it must be configured to load the main '.js' script file. It can be done by right clicking on the blank SMP panel:

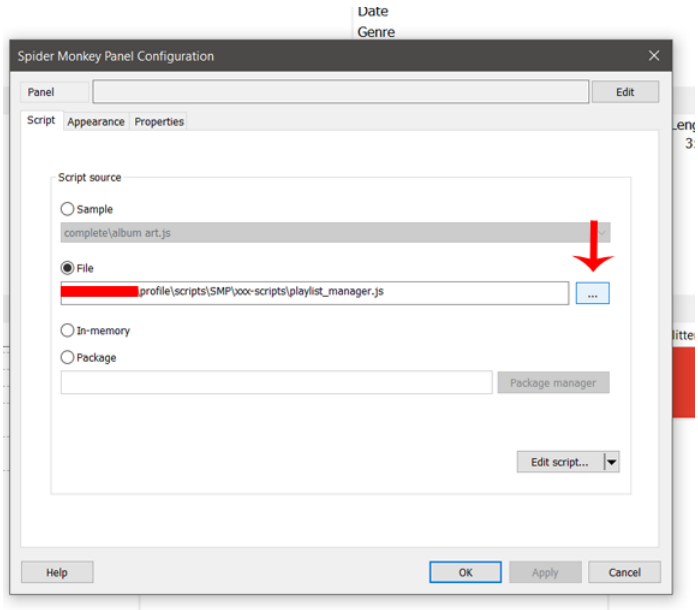


Figure 7: Loading the **'playlist_manager.js'** script within a SMP panel.

6 Wine installations

Script is 100% compatible with unix systems although some points should be taken into consideration:

- 'Segoe UI' must be present, as it is the font by default on the script's UI.
- In case the font is not found (check console log), revise your default font on Wine. Changing it to a different one than 'Segoe UI' may give problems or crashes in some cases.
- [Additional] required fonts must be installed to avoid crashes. Is not only eyecandy.
- At some point the script may use external CMD tools which must be compatible with 32 bit systems in Wine (since that's the most popular version, with IE). In this case 32 bit binaries for 7z and CmdUtils are bundled.
- Script may throw an error after installation (either a foobar crash or panel crash) due to 'missing files'. There is a fix for that, check [XI] and [XII]. It's a SMP bug, don't report it.
- Wine installations are specially sensible to missing files, wrong paths, etc. which may lead to crashes or bugs usually not found on Windows installations. Have that in mind and double check your installation procedure before reporting an error [XII].
- All scripts try to use wine-friendly methods, focusing on configuration settings that can be changed via menus or the UI panel, instead of using HTML (which only works on Windows), known working fonts, etc.
- Please read SMP Wine page and feel free to report (me) any additional problem with these scripts.
- Read Wine foobar thread for more info and tips.

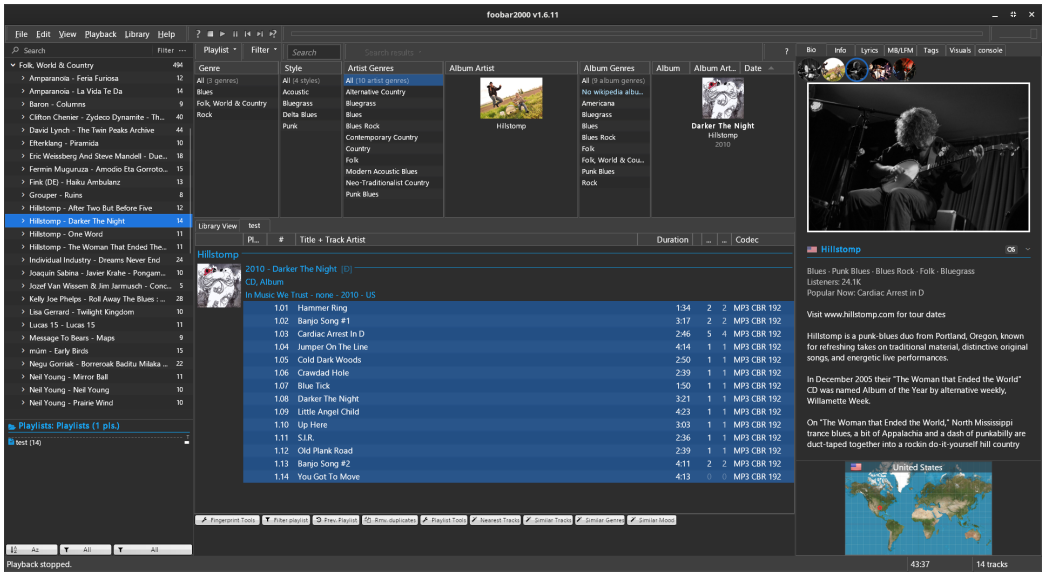


Figure 8: Script working on Wine along Playlist Tools and World Map.

Part III

Features

7 Tracking folder

The key feature of a file manager, whether it's a library or playlist manager, can be reduced to tracking paths in some way. The playlist manager tracks one folder per instance[9] and lists all readable playlist files found on it. Auto-playlists, while not being "on the folder" are considered linked to it, so different playlist manager instances have different Auto-playlists associated. The tracked folder can be set on the header menu [18]:

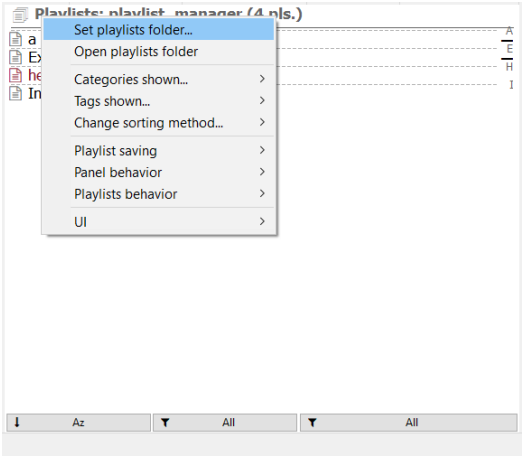


Figure 9: Setting tracking folder.

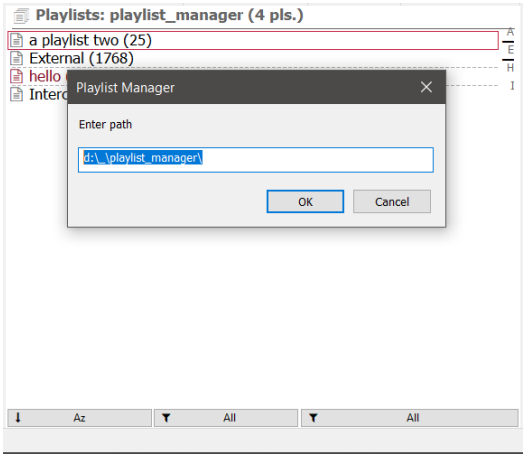


Figure 10: Tracking an absolute path.

Sub-folders are not tracked[10] and the tracked path is meant to be set once and done, so categorization is done with metadata instead [22.3]. Why? It offers the same functionality ('virtual sub-folders') while not making it unnecessarily complex... at the end these are playlist files (links to tracks), not tracks. So they should be physically -more or less- all on the same place while also having some kind of metadata to easily categorize them[11].

The path may be an absolute or relative path. Relative paths have their root set at the Foobar2000 installation path (where the '.exe' resides in). So something like '.\profile\playlist_managerís perfectly fine. There are multiple reasons to prefer relative paths, check the tips section for more info [41].

[9] Yes, that means you may have multiple manager panels.
[10] i.e. If 'Playlists' is tracked, 'Playlists\Summer' will not be tracked too'
[11] If you think otherwise, nothing stops you to create multiple sub-folders and having multiple manager instances in a panel with tabs for easy access.

8 Managing Playlist files and Auto-playlists

8.1 Auto-playlists

Contains all functionality on Auto-playlist Manager by marc2003 plus more:

- Create, rename, delete Auto-playlists.
- Edit query, sort pattern and sort forcing.
- Adds tooltip info, UI features, filters, etc.
- Number of tracks output is updated at foobar startup, 'Manual refresh' [17], when loaded or automatically at startup.
- Queries and sort patterns are checked for validity before using them, instead of crashing.
- Import playlists from Auto-playlist Manager by marc2003[13.1].
- ...

Auto-playlists are essentially treated the same than standard playlists, with their physical file being a json formatted file (containing all Auto-playlists). They can be exported or imported the same than standard playlists, cloned as standard playlists, etc. In other words, the differences are in their internal format and their set of features associated, but that's all[12]. There is no differences in the way they are managed or presented to the user.

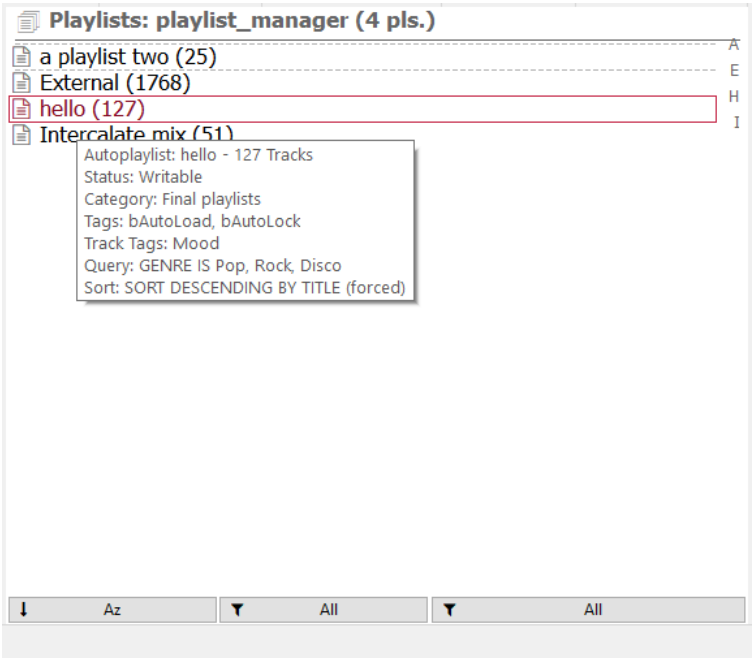


Figure 11: Tooltip shows all relevant info, like query, sort pattern, etc.

[12]Obviously standard playlist have no query or sort patterns. But whenever a tool or feature make sense in both types of playlists, it's implemented on both.

8.2 Standard playlists

Standard playlists are the common playlists used within Foobar2000, the ones where you may add, remove or reorder tracks (contrary to Auto-playlists which are query-generated). When a playlist loaded in Foobar2000 is also tracked by the manager, then a physical playlist file within the tracked folder is associated to it. That process is called 'playlist binding' [8.7]. There is nothing special about it, it's simply a way to say the physical file and the playlist within foobar are in sync.

Playlist files may be freely created, renamed, deleted, etc. and their associated Foobar2000's playlist counterpart will follow the same changes (if desired). The same applies in both directions (with some logical exceptions). It's key to understand this; there is a real physical playlist file with all those tracks written to it: If at some point you close a playlist within Foobar2000 and restart it, that playlist is gone for good[13]. On the other hand, if you do the same with a playlist from the manager, you may simply reload the file. The physical file is never deleted unless you do it on purpose, so you can always load it at any point no matter what you do with the playlist on the tabs.

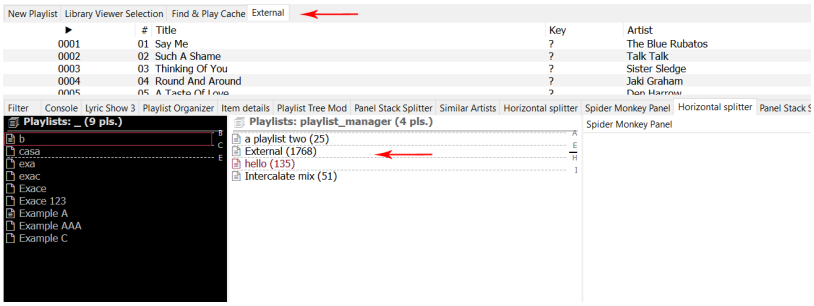


Figure 12: The playlist tabs have 4 playlists currently loaded, while the manager has other playlists which are not loaded yet.

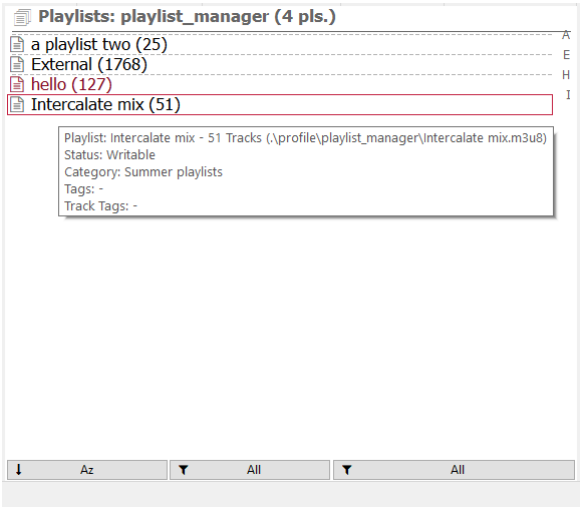


Figure 13: Playlist's tooltip show essentially the same info than Auto-playlist's one. Note both types of playlists are easily identified by their color code (configurable).

[13] Because playlists only reside in the program as long as they are loaded within the UI.

8.3 How paths are written: absolute and relative paths

In writable playlists formats, tracks may be written as absolute or relative paths (considering as root the folder where the playlist resides in). For ex:

Absolute path:
D:\Music\Big Retro Hits 90s\007. Chateau Pop - Maniac.mp3

Relative path on current root (D:\):
.\Music\Big Retro Hits 90s\007. Chateau Pop - Maniac.mp3

Relative path one level up from root (D:\Playlists\):
..\Music\Big Retro Hits 90s\007. Chateau Pop - Maniac.mp3

```
[D:]
├─ [Playlist]
│    └─ My playlist.m3u8
└─ [Music]
   ├─ [Big Retro Hits 90s]
   │  ├─ 007. Chateau Pop - Maniac.mp3
   │  └─ ...
   └─ ...
```

Relative paths may be enabled changing the related configuration on the header menu [18]:

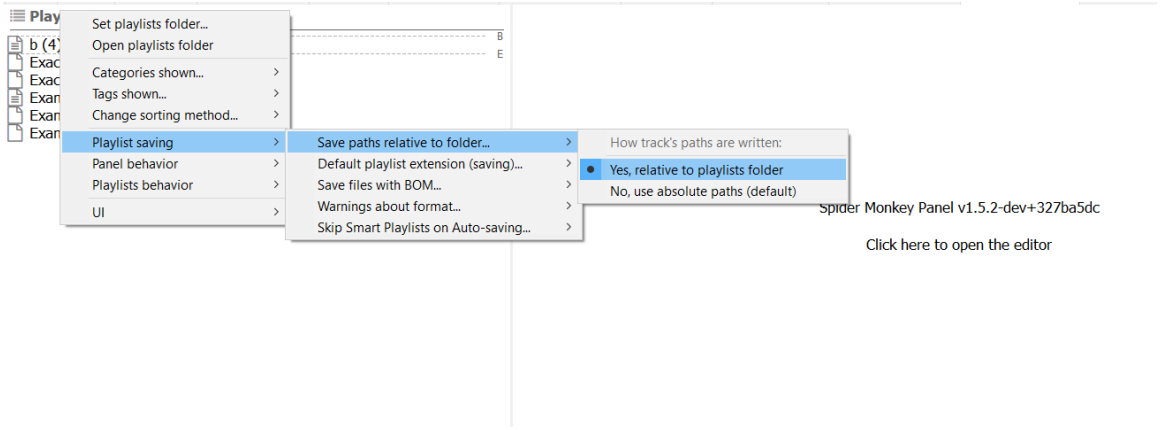


Figure 14: Enabling relative paths for tracks on (new) playlists.

Note enabling this feature is not enough condition per se, since the tracks must reside in the same drive disk to have relative paths working. Whenever relative paths can not be set, absolute paths are used as fallback. Relative paths would not be available on the following example :

```
[D:]
├─ [Playlist]
│    └─ My playlist.m3u8
[E:]
├─ [Music]
   ├─ [Big Retro Hits 90s]
   │  └─ 007. Chateau Pop - Maniac.mp3
```

8.4 Setting playlists format

All playlists created or saved by the manager use the format (extension) set on the panel, no matter their original format -by default-. In other words, new playlists will use that format, but also any external playlist added to the tracked folder will be converted to the set format on auto-saving unless manually locked to avoid so[8.9]. The default format can be changed at the header menu [18], along the default behavior of forcing it to existing playlists instead of maintaining their original format.

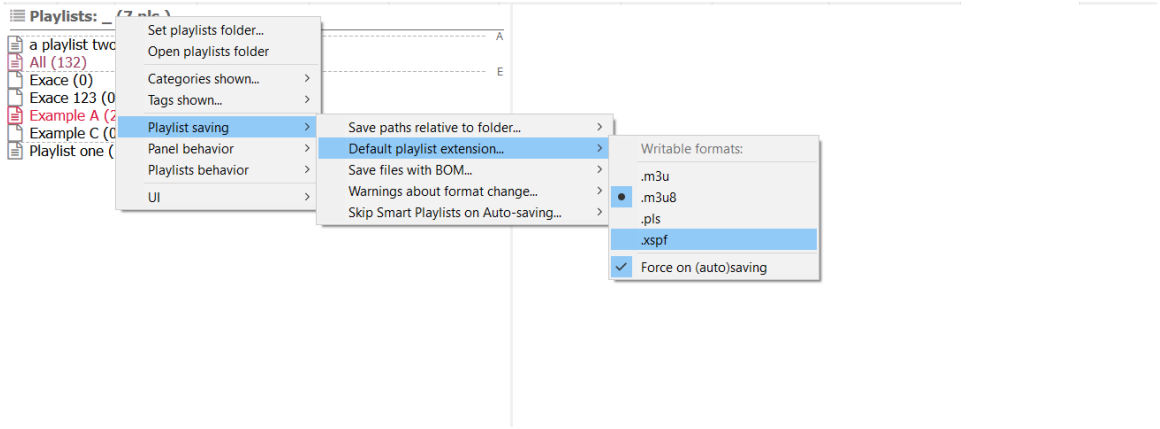


Figure 15: Playlists default format.

Additionally, playlist files may be written with or without BOM (files are always UTF-8 encoded) [VIII]. For compatibility purposes it can also be enabled or disabled on the header menu [18]:

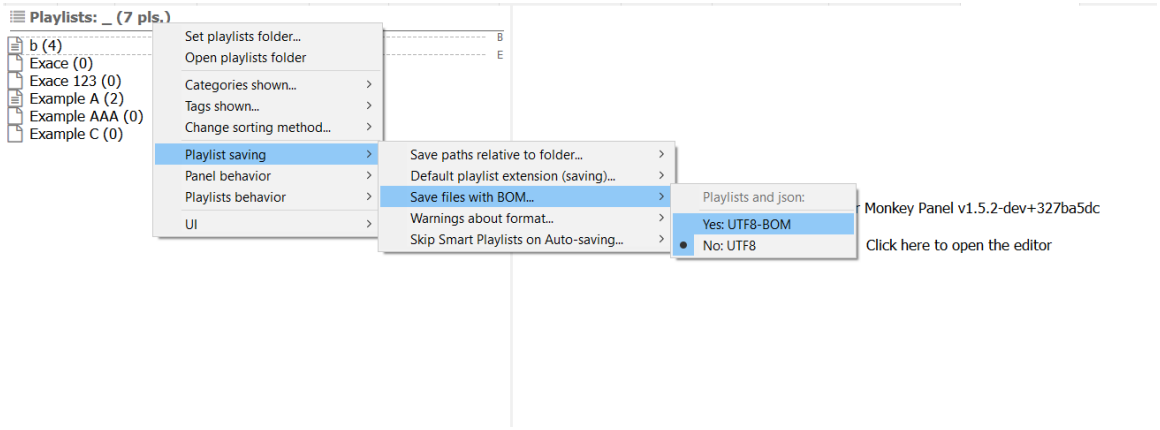


Figure 16: BOM configuration.

Further information about the differences of the multiple formats available and their structure can be found on the respective section [V]. Actions which involve editing playlists without changing the format to the default one are named **format friendly actions** [VIII].

8.5 Creating playlists

Creating new playlists [files] may be done easily in 2 ways on the list contextual menu[18]: either an empty playlist or creating a new file from the active playlist ('cloning it'). The first set of options simply creates an empty playlist file on the tracked folder and then also a new playlist on Foobar2000's UI with the same name:

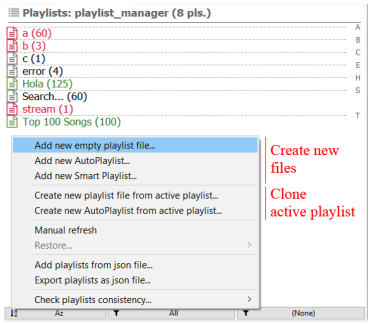


Figure 17: Menu entries to create playlists. A popup will appear to input the name.

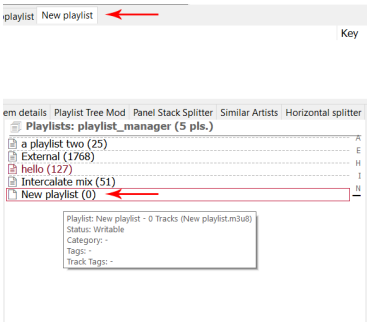


Figure 18: Playlist is created in both places: the UI and the physical folder.

The other options create the physical[14] file in the tracked folder, bound to the active playlist. A popup will appear asking to maintain the name (thus using the active playlist) or input another one (creating and using a clone of the active playlist with the new name).

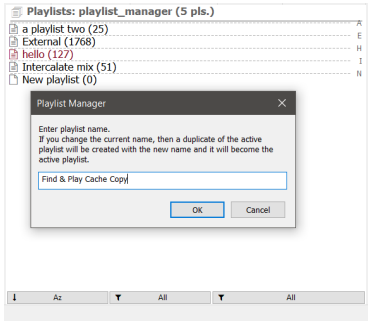


Figure 19: Cloning active playlist.

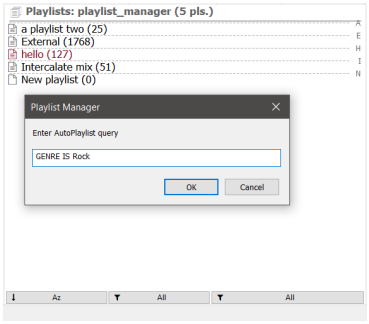


Figure 20: Query input for Auto-playlist.

Auto-playlists and Smart Playlists are also created using the same contextual menu. Apart from the name, query and sorting are also set via popups.[15]

New playlists will inherit the category or set of tags [9] from the current view filters [22], i.e. if the current view only shows a single category, then all new playlists created will be set with such category by default. On the contrary, since playlists may have multiple tags [27.2], all from the current view will be added by default (only when current filter excludes playlists without tags).

[14] The format used is the one set at configuration.

[15] The entry to create an Auto-playlist from active playlist is not available unless the active playlist is also an Auto-playlist.

8.6 Playlist loading *-format friendly action-*

Foobar2000 loads playlists pretty fast thanks to using a binary playlist format (.fpl) instead of looking for the physical track files. The binary format stores all the relevant metadata needed to then display the tracks within foobar200.

On the contrary, loading any of the writable format playlists in native Foobar2000 is really slow. The physical files are loaded one by one and then their metadata retrieved... that process is done asynchronously and can easily take minutes as soon as a playlist has more than a hundred of tracks[16].

The manager uses those writable formats to create clones of the playlists within foobar but they are loaded as fast as the native binary format (.fpl) by finding matches on library for every track. Since playlists are supposed to be pointing to items already on Foobar2000's library on most cases, caching the paths of every item on library greatly speeds up the process[17].

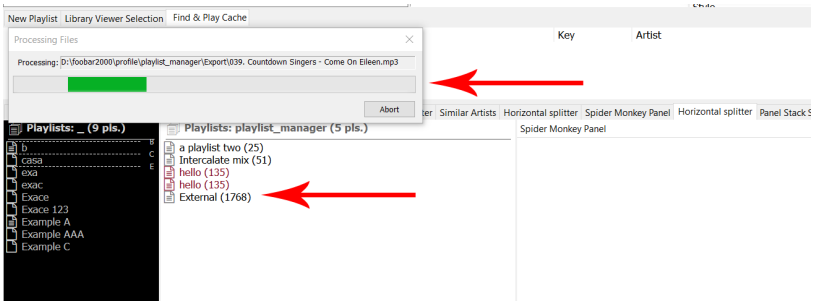


Figure 21: Async loading of a playlist file with tracks not present on library.

Having in mind the previous comments, here are some key rules to optimize the loading time of playlists:

- Use the playlist files to store only tracks present on the library whenever it's possible.
- Don't mix external items with items on library on the same playlist.
- Replace dead items whenever it's possible.
- Add streams to their own playlists.
- Add external items to their own playlists.
- Use the tools to check for external and dead items [16.2] and avoid unoptimized playlists.

[16] Note subsequent loading of the same playlist is much faster since those items have been already cached.
[17] This erratic behavior has been already reported at Foobar2000 support forums without an answer.

8.7 Playlist binding

Binding is the action of associating a playlist within Foobar2000 and a physical file for syncing purposes. This is done by name, so a playlist named 'ABC' in the manager will be a mirror of a playlist named 'ABC' in Foobar2000 UI. Additional ways to handle playlist names can be set using UUIDs [27.5].

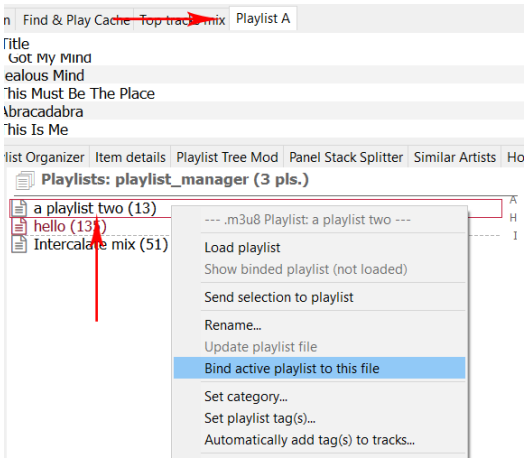


Figure 22: Bind selected playlist to active playlist.

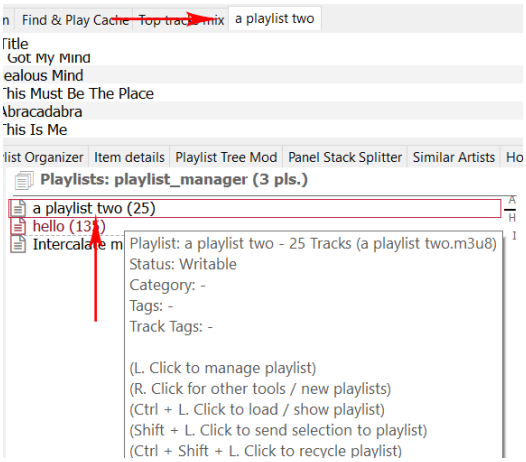


Figure 23: The active playlists is renamed and the playlist file updated with its contents.

Bound playlists are auto-saved if such feature is enabled, i.e. any change made to the Foobar2000 playlist will be automatically reflected in the physical file. Some restrictions may apply though[18]. Other manual actions include:

- Reload: reloads the playlist within Foobar2000, overwriting any non saved changes.
- Update / Force update: saves any change to the physical file[19].
- Delete: deleting the file also asks to delete the bound playlist within Foobar2000.
- Rename: renaming the file also renames the bound playlist[20].
- Show bound playlist: bound playlist within Foobar2000 becomes active playlist[21].

Since the only way to assign a playlist file to a Foobar2000 playlist is forcing both to have the same name, a new problem appears... what about duplicates? By default the Playlist Manager will not allow 2 playlists with the same name if there is already a playlist file named equal to them. i.e. no duplicates allowed for tracked playlists. Note there is not an [exposed] UUID associated to every playlists to work with, so in fact the only thing that may be used as UUIDs are the names. There are multiple configurable UUIDs that can be set instead of the plain name that can be used to allow some kind of 'duplicates' or to differentiate tracked from non tracked playlists [27.5].

[18] Playlist may be locked for changes, a non writable format may be used, etc.
[19] The manual counterpart of auto-saving.
[20] This is a requisite, since the link is the name!
[21] Like the 'Show now playing' action, but instead it simply shows the selected playlist.

8.8 Deleting and restoring files *-format friendly action-*

Playlist files deleted within the manager context are not permanently deleted but sent to the Recycle Bin. Timestamps are used to uniquely identify files; this is done to ensure no collisions with other files within the Recycle Bin. Manually deleting a playlist using the playlist contextual menu [18] allows restoring at a later point using the list contextual menu[22].

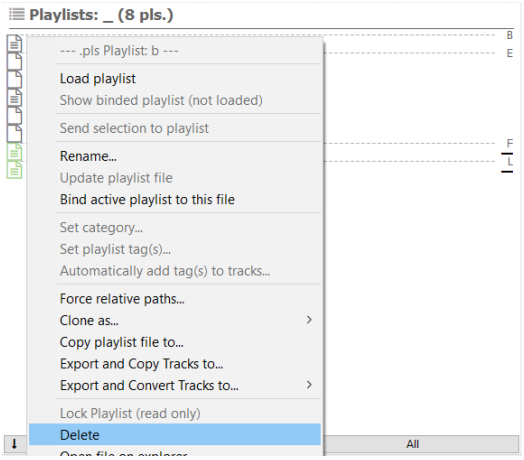


Figure 24: Delete selected playlist.

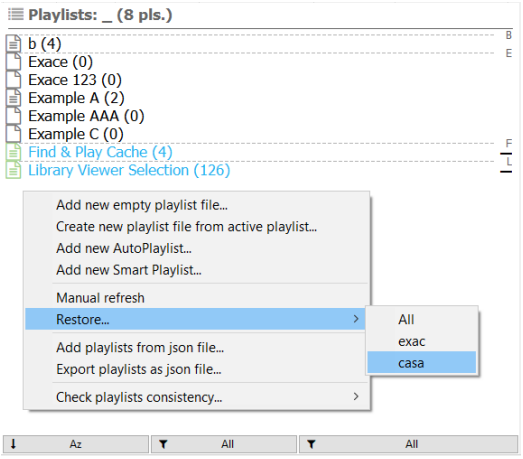


Figure 25: Restore deleted playlists.

Additionally, a backup of the Auto-playlist/fpl json database is created every time the panel is loaded and previous backups are sent to recycle bin[23].

Since playlist files are sent to the Recycle Bin every time an edit is done, as long as the Bin is not emptied, previous versions of the playlists may be found indefinitely. Reverting an unwanted edit, file corruption or restoring files deleted long time ago is as easy as restoring a previous version.

[22] Alternatively, the file may be found on the Recycle Bin... so it could be restored manually after stripping the timestamp.

[23] In other words, there is always 2 versions of the file. The current and the previous [start-up] one.

8.9 Locking files *-format friendly action-*

Playlist may be locked to disable editing, overwriting the physical file or any change apart from unlocking or renaming it[24]. Locked playlists are also skipped on auto-saving.

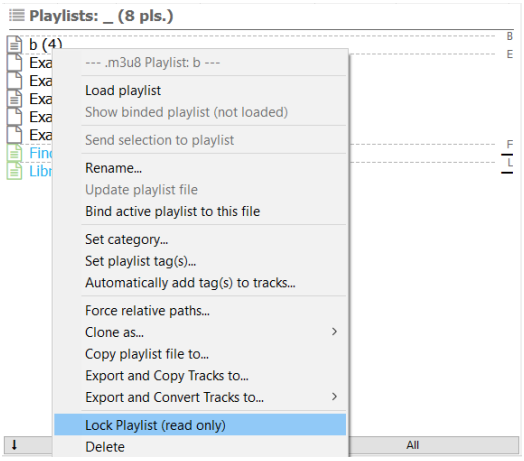


Figure 26: Lock selected playlist.

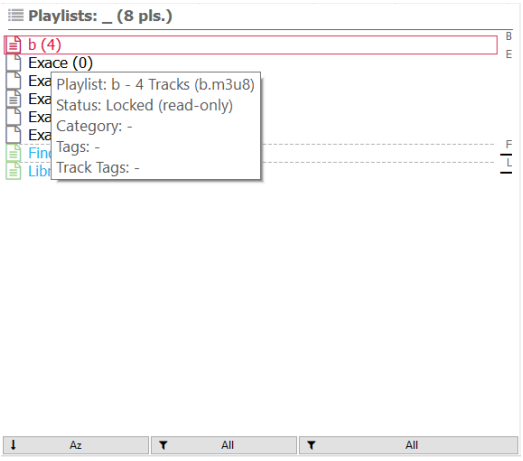


Figure 27: Locked playlist tooltip.

Note a loaded playlist within foobar may be edited even if its associated physical file is locked. This is done on purpose [25], to allow playlist edits while having the physical file locked and untouched. At any point the playlist may be unlocked and changes be saved or it may be directly forced to update the changes. Alternatively they may be discarded simply reloading the playlist file.

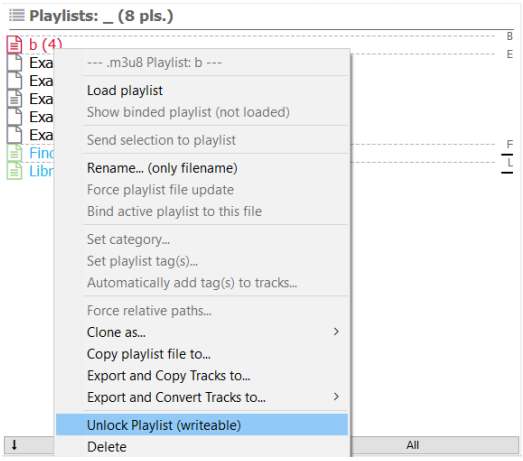


Figure 28: Unlock selected playlist.

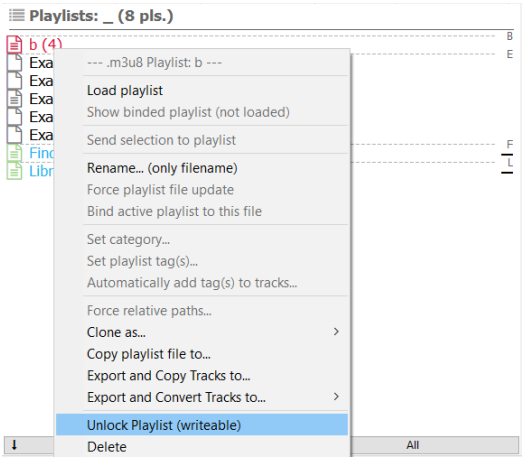


Figure 29: Force saving to locked playlist.

Native foobar playlists files (.fpl) are locked by default to avoid changing their format[26].

[24] Only the physical file is renamed in such case.

[25] Contrary to what other Foobar2000 playlist managers do, which lock/unlock the playlists within the program (since there is no physical files).

[26] Configurable at properties panel only.

8.10 Editing UI playlists locks *-format friendly action-*

The manager may also be used to lock/unlock UI-only playlists or playlists already loaded on UI (leaving their associated files alone). Lock/unlock actions don't work like a simple switch but in a fine grained way, i.e. different foobar playlists actions may be locked instead of just locking all or nothing. Available actions are called in this context "locks". An Auto-playlist is an example of this: their sort order may be changed even if the items can not be removed/added manually.

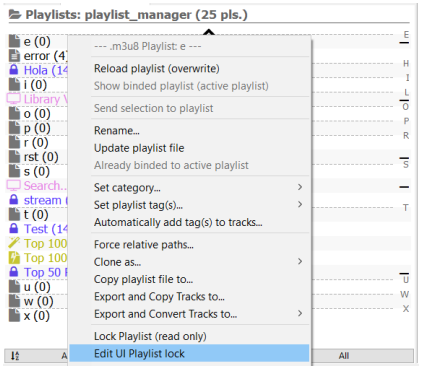


Figure 30: Menu entry on loaded playlist.

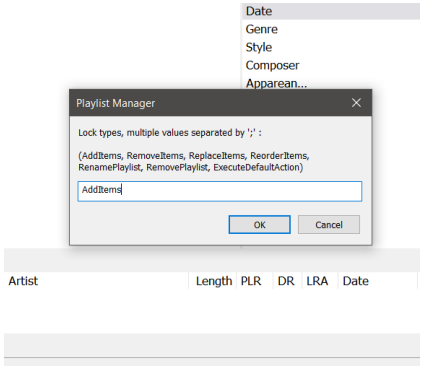


Figure 31: Lock edit popup.

List of vailable locks (whose actions should be self explanatory):

- AddItems
- RemoveItems
- ReorderItems
- ReplaceItems
- RenamePlaylist
- RemovePlaylist
- ExecuteDefaultAction

Note lock edit is only available for playlists which are not being currently locked by other plugins or Foobar2000 itself. For ex. an Auto-playlist can not be edited since the locks are imposed by the program. There are other plugins which may add special locked playlists too. In such case the option will be greyed out. Locks' parent may be identified on the menu entry or the status bar. Therefore lock edit is only avaiable when it has been added by SMP.

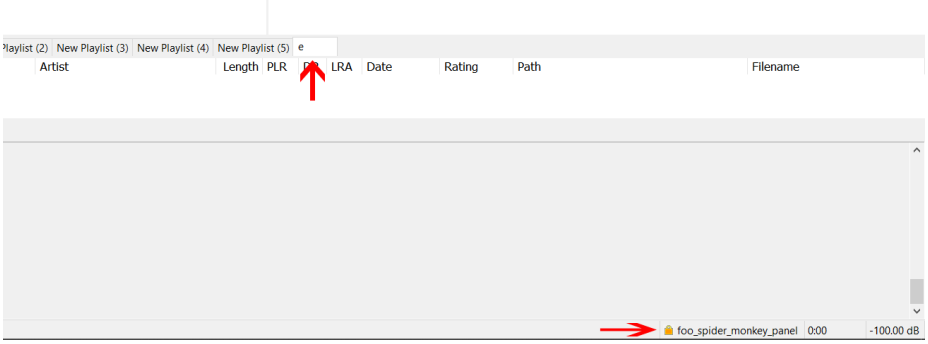


Figure 32: Lock status is displayed at the status bar on CUI.

9 Working with playlist metadata

Playlists may have multiple metadata associated [27], in particular tags (multiple allowed), category (one allowed) and other variables related to special actions [11] [12].

Categories may be used for easy playlist classification since playlist are only allowed to have one category -like virtual folders-. They are also shown on the tooltip over a playlist [22.8], but contrary to tags, categories can be used to permanently filter the list, cycle through them, etc. Categories can be set on the selected playlist contextual menu [18].

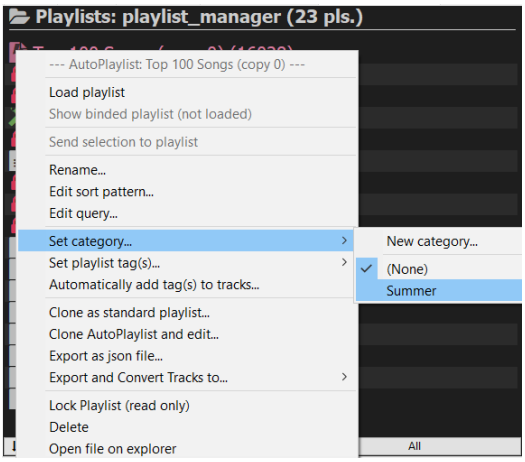


Figure 33: Setting category menu entry.

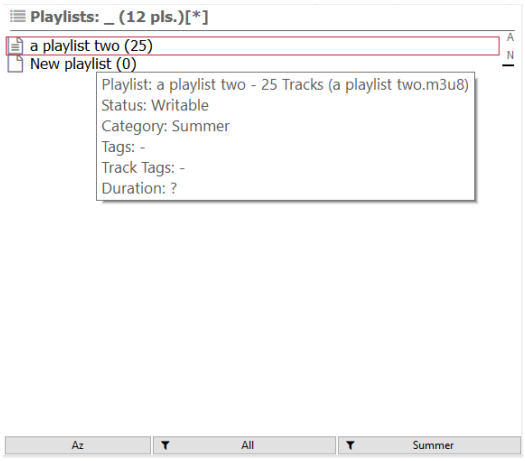


Figure 34: Category shown on tooltip.

Tags may be used for informative purposes since they will be shown on the tooltip over a playlist [22.8]. They have no further purpose, although may be used temporarily to filter the current view too [22.5]. Tags can be set on the selected playlist contextual menu [18].

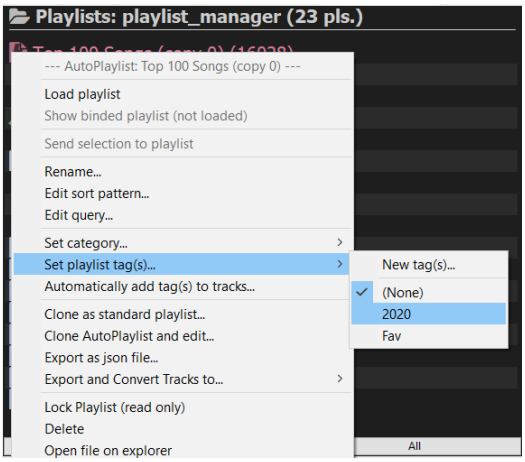


Figure 35: Setting tag menu entry.

Figure 36: Tags shown on tooltip.

9.1 Metadata inheritance

To simplify working with tags and categories, metadata inheritance has been added related to the current filter view [8.5] or the original source used (cloning) [13.3].

For example, while using category cycling [22.4] or tag filtering [22.5], any new playlist created will have that category or tags set by default... thus saving time on later editing [8.5]. In real usage it's natural to expect that a new playlist should be shown on the current view, so if you are currently working with 'Summer' playlists category, any new playlist should be added to that virtual folder too. Otherwise, since the current view is being filtered, the new playlist would be hidden due to not having a category by default.

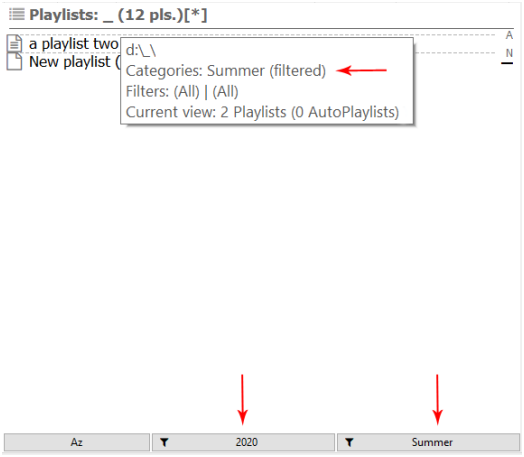


Figure 37: Current view filtered in tag and category.

Figure 38: New playlist inherits the tag(s) and category.

Since playlist can only have one category, multiple categories can not be inherited at the same time... thus only category cycling [22.4] provokes inheriting, or filtering all but one category [22.3]. If multiple categories are shown at the same time, then new playlists will not have one set by default[27]. On the contrary, multiple tags can be inherited at the same time, so all currently being shown on the list view will be used unless files without tags are not filtered[28].

Note blank playlists, with no metatada, should be created disabling all category/tag filtering [22.7] first. Otherwise they may inherit metadata from the current filtered view.

[27] This is to be expected! Which one should be added and why?

[28] In other words, if the current view show '2020' tagged playlists but also those without tags, no inheritation is applied... since untagged playlists are not being filtered. This is again following some natural logic: if you only want an specific tag, you would filter the list to show only that... in such case new playlists would fit there. But the same would not apply if untagged playlists are shown. Should the new playlist have a tag or none?

Cloning a standard playlist [13.3] also forces metadata inheritance, so it can be used as an easy way to create new playlists with existing metadata: just delete all its tracks to get a blank playlist ready to be used. Cloned Auto-playlists follow the same behavior, whether they are cloned into a new Auto-playlist or a standard playlist.

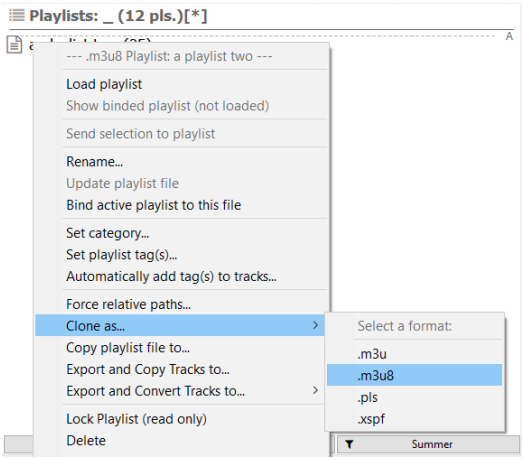


Figure 39: Cloning menu entry.

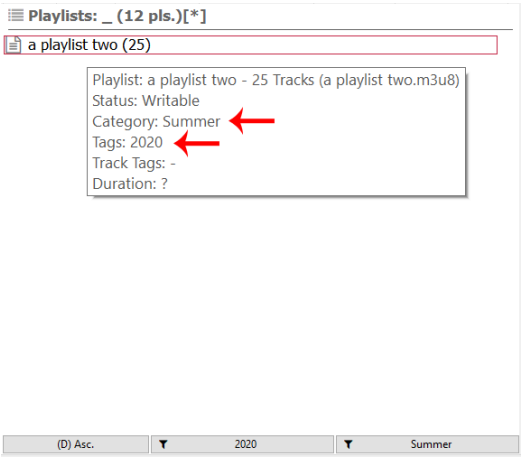


Figure 40: Source playlist metadata.

Figure 41: New playlist metadata.

10 Auto-saving, Auto-loading and Auto-backup

10.1 Auto-saving

The first refers to the capability of duplicating any change made within foobar loaded playlists (usually those on the playlist tabs) to their physical file counterpart. Obviously, only those playlists with a bind file in the Playlist Manager panel will be tracked for changes[29]. This may be configured, on the header menu [18], changing the tracking interval (ms) or completely disabled (to copy back changes manually on demand).

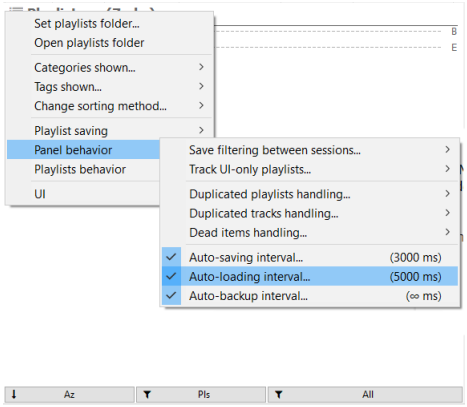


Figure 42: Auto-save configuration.

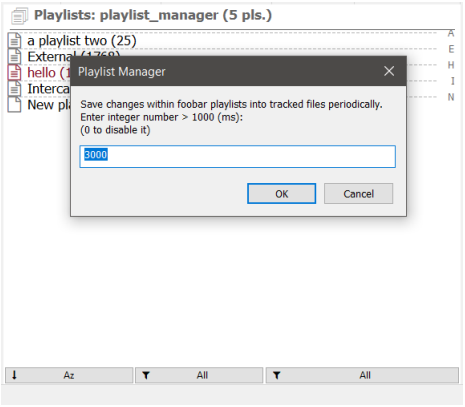


Figure 43: Auto-save interval in ms.

The manager will also throw a popup whenever a saving action would change the file format of a playlist, allowing to abort it. Such warning can be configured on the header menu [18]. Smart Playlists [35] are also skipped on auto-saving by default.

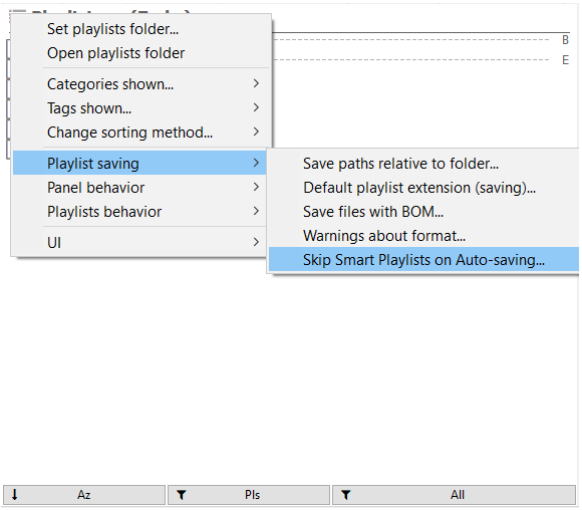


Figure 44: Format changing warnings and Smart Playlists saving.

[29] In other words, it may be possible to have both tracked and non tracked playlists within foobar.

10.2 Auto-loading *-format friendly action-*

The second refers to the capability of automatically tracking changes within the tracked folder, reflecting any change made to files on real time. This may be configured changing the tracking interval (ms) or completely disabling it (to only reflect changes manually or on every startup).

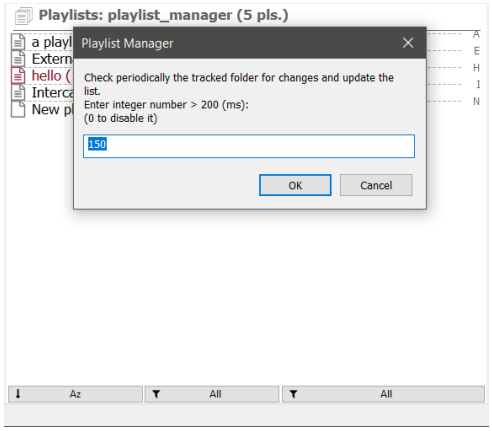


Figure 45: Auto-load interval in ms.

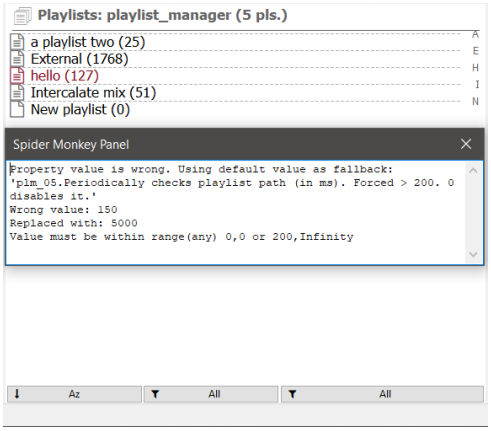


Figure 46: Wrong values trigger a warning.

10.3 Auto-backup *-format friendly action-*

The former refers to the capability of automatically perform backups -zip files- of playlist files [V] within the tracked folder ('*_backup*' subfolder). It may be fully disabled (0), performed according to an interval (ms) or only whenever a playlist is loaded and at foobar shutdown (infinity). Files are named following date ISO format.

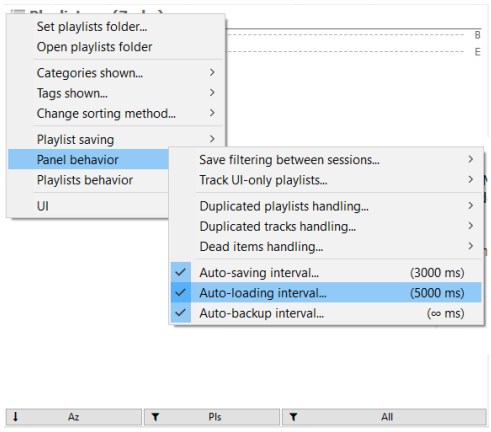


Figure 47: Auto-backup interval in ms.

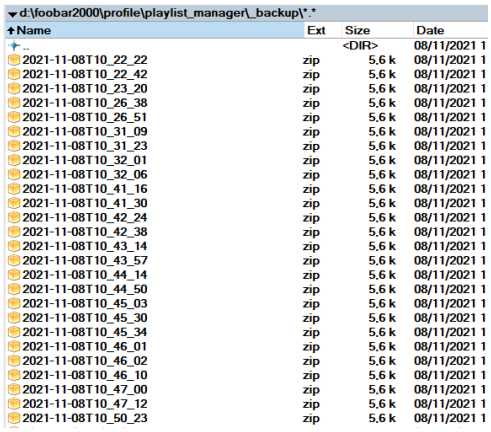


Figure 48: Backup files.

11 Automatic playlist actions *-format friendly action-*

Some reserved playlists tags names are used for special purposes by the manager to automatically perform some actions as soon as it loads a playlist with such keywords:

- 'bAutoLoad' makes the playlist to be loaded within foobar automatically (on the UI). Meant to be used on remote servers with online controllers.
- 'bAutoLock' locks the playlist as soon as it's loaded on the panel.
- 'bMultMenu' associates playlist to dynamic main menu entries [19] which are applied on batch to a group of playlists.

The feature must be explicitly enabled on the header menu [18] to work; i.e. a playlist with such tags will not automatically perform any action until enabled globally.

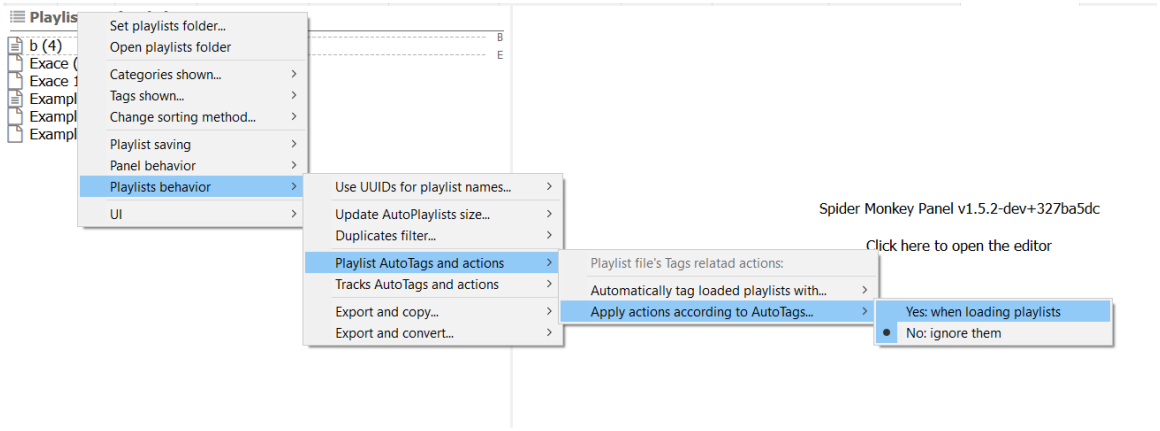


Figure 49: Enabling Automatic playlist actions on the header menu.

Additionally, tags may be added to playlists automatically as soon as they are loaded. This may be used to enforce an specific tag on all playlists tracked by the manager, no matter what it's set on the playlist file. Furthermore, used along the automatic playlist actions, it may be used to force loading or locking of all tracked playlist [30].

[30] It's disabled by default, so this allows both: to selectively apply actions to 'tagged' playlists or apply them to all.

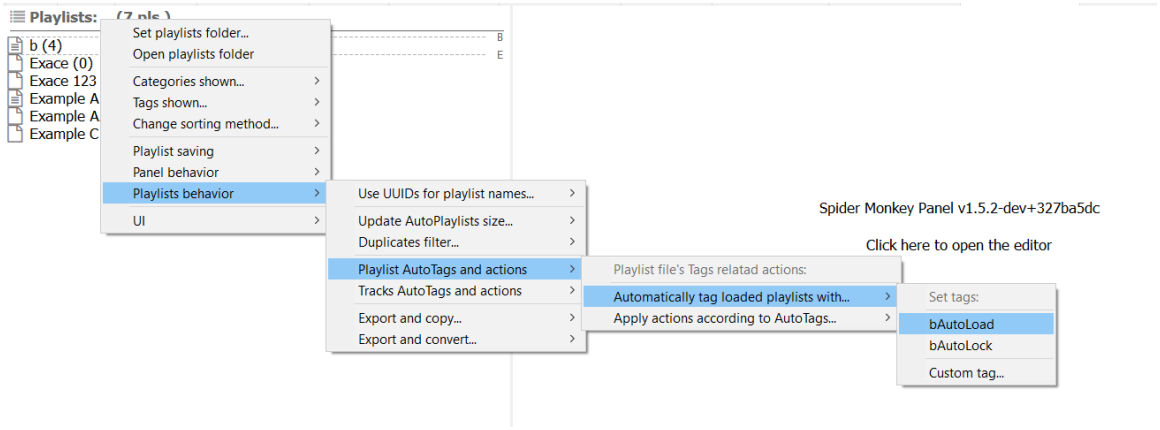


Figure 50: Setting tags to be added automatically to tracked playlists.

Obviously, nothing stops you to use it to simply tag playlists with a custom tag that has nothing to do with automatic actions. In any case, the tooltip shows the playlist tags (whether they are for informative purpose or used for actions):

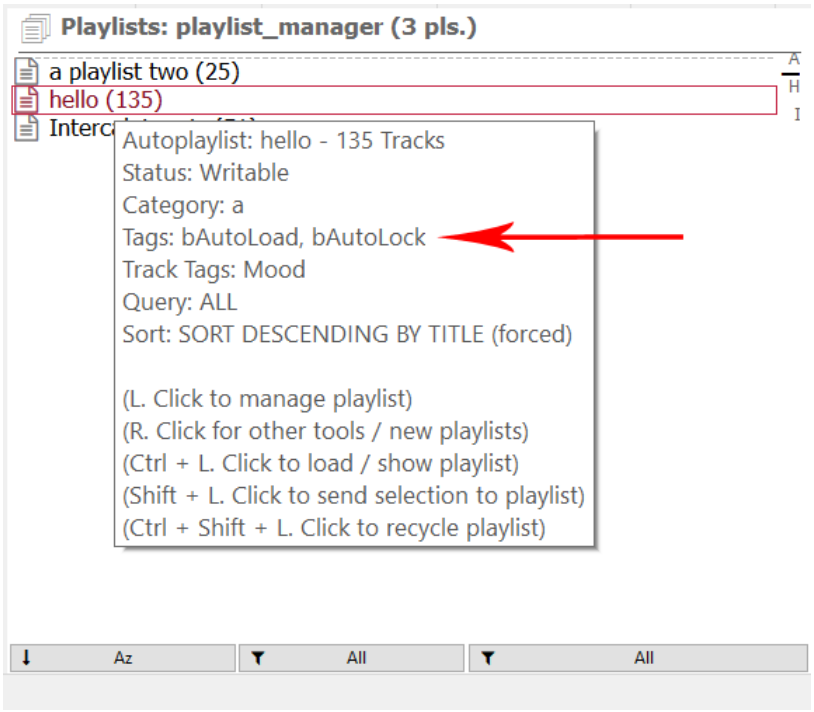


Figure 51: [Playlist] tags on tooltip.

12 Automatic track tagging *-format friendly action-*

Playlists may be used to automatically tag tracks on demand (the moment you add a track) or on startup [27.3]. The conditions to tag tracks on a playlist are set using the contextual menu for the selected playlist [18] and must follow json format [VIII]. For example:

Example	Description of how tracks would be tagged
[{"rating":5}]	%rating% and a value of 5
[{"mood":"Chill"}]	%mood% and a value of 'Chill'
[{"year":"$year(%date%)"}]	%year% and a year value from the full date tag
[{"checked":"JS:todayDate)"}]	%checked% and a date value using JavaScript.

Table 1: Automatic track tagging examples.

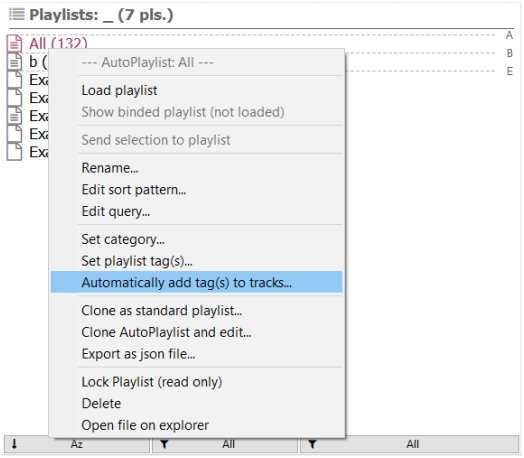


Figure 52: Add track tags to selected playlist.

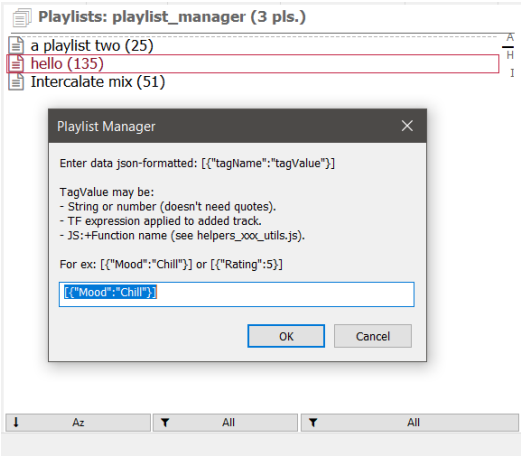


Figure 53: Track tags popup.

As noted, the use of arbitrary JavaScript functions is allowed, but they must be defined at a helper file '.\helpers\helpers_xxx_utils.js'. In this case, the function it simply returns the date Y-M-D-h-m-s in which was tagged. Users may add their own functions to it.

Playlists may have multiple track tags, in that case all of them would be applied to the tracks when required. Other features include:

- Can be configured separately for standard playlists, Auto-playlists, locked playlists and individual playlists.
- Standard playlists may be used to easily tag your library just by sending them to the right playlist (which don't need to be loaded at all).
- Auto-playlists Auto-tagging allows to automatically (and periodically) apply some tagging logic to the current library following some condition.
- Allows multiple conditions (must follow json format) [VIII]. Look at playlist metadata for more info 27.3.

The feature must be explicitly enabled on the header menu [18] to work; i.e. a playlist with track tags will not apply them until globally enabled.

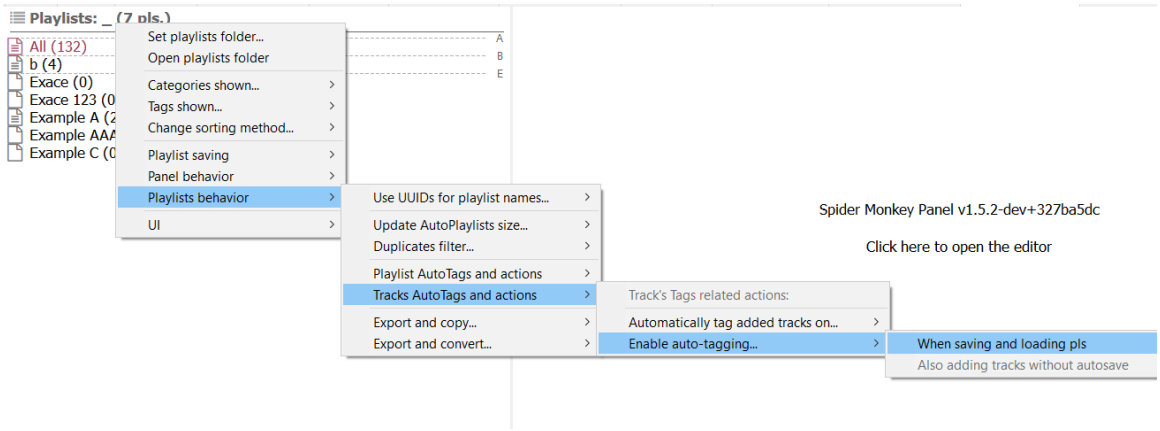


Figure 54: Enabling Automatic track tagging on the header menu.

When a playlist has track tags set, the tooltip shows the tags which would be written in case the feature is enabled:

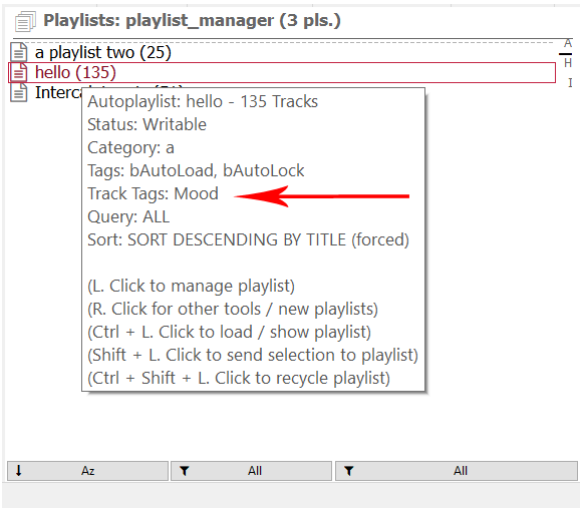


Figure 55: Track tags on tooltip.

Enabling Automatic track tagging for Auto-playlists at startup involves processing their queries and thus requires some processing time. The process is done asynchronously on the background [VIII], thus not requiring additional startup time in most cases, but enabling the option should be considered carefully on resource constrained PCs. In any case, **it's recommended to make a moderate use of this feature**; note having multiple Auto-playlists trying to tag the same files at the same time may lead to errors (due to files being blocked), your mileage may vary.

13 Exporting Auto-playlist

13.1 Exporting or importing Auto-playlist files

The original idea of a playlist manager found on Auto-playlist Manager by marc2003 consisted only on a list of Auto-playlists which could be loaded on demand... thus to make it easy to transfer all those Auto-playlist to this manager there is an option to directly import its json files[31] on the list contextual menu [18].

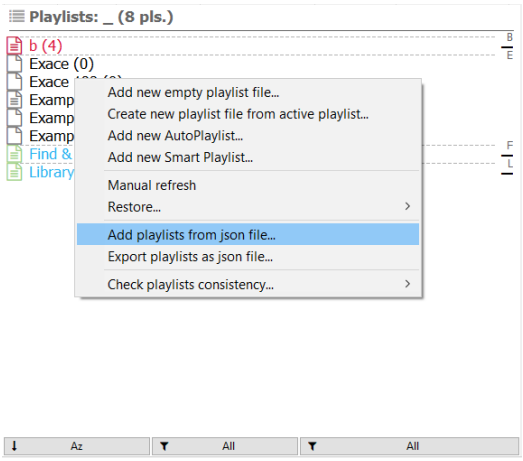


Figure 56: Import Auto-playlists from json.

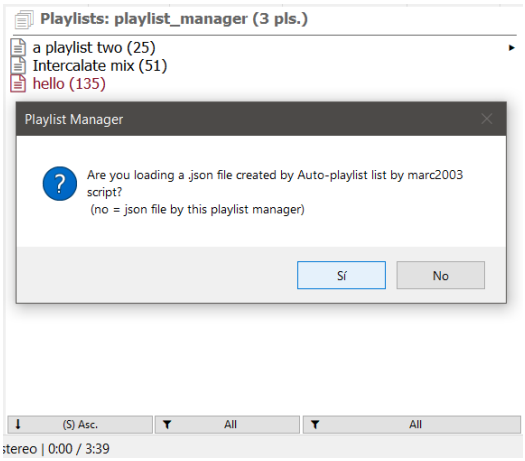


Figure 57: Popup asking about json's source.

At the importing process 2 options may be chosen depending on the json file being originary from marc2003's panel or from this one[32]. All Auto-playlists found will be checked for validity and added to the current playlist list on the manager.

Auto-playlists exporting process for this manager is equivalent to marc2003's one, just copy/paste the appropriate json file[33]. To import it at another panel instance just follow the steps written previously, and choose the appropriate option (files from this panel).

Note the json file from this manager contains both Auto-playlists and .fpl virtual playlists for metadata purposes [33], but the latter are discarded when importing using the menus as described[34]. Anyway the format from this manager is not backwards compatible with marc2003's script, so it doesn't affect in any way for regular users.

Alternatively, Auto-playlists from the panel may be selectively exported using the playlist contextual menu option [13.2].

[31] marc2003's json file (at foobar profile folder) will be at '.\js_data\autoplaylists..json'.

[32] Since marc2003's panel follows its own schema for Auto-playlists, some internal conversion is needed [34]

[33] For ex. if the panel is set to track 'H:\My Music\Playlists', then the playlist json file (at foobar profile folder) will be at '.\js_data\playlistManager_Playlists.json'.

[34] Contrary to marc2003's script, whose json file only has Auto-playlists.

13.2 Export as json file

Single Auto-playlists may be exported as json files, instead of following the general procedure (which exports all of them at once), using the selected playlist contextual menu [18]:

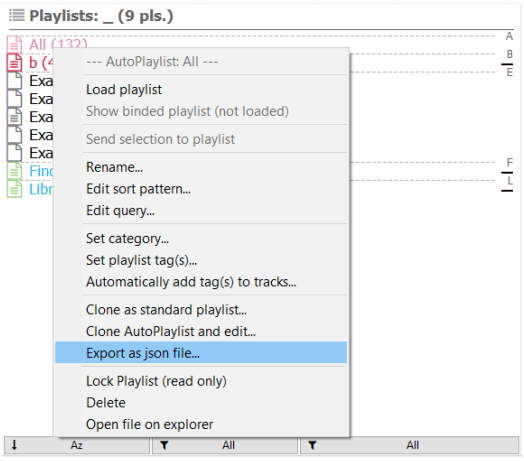


Figure 58: Export selected Auto-playlist.

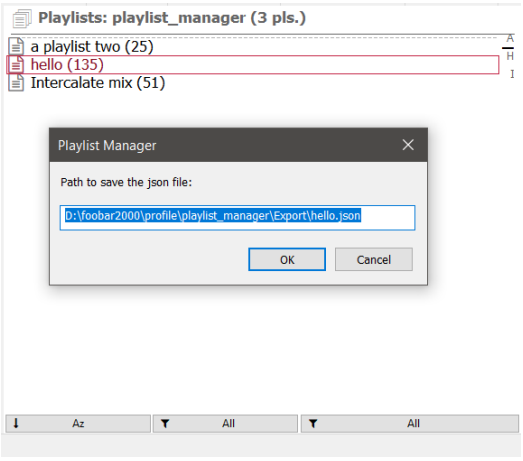


Figure 59: Path to exported json file.

Alternatively all Auto-playlists may be exported at once using the list contextual menu too. This option has an advantage over the general procedure of just copying the json file: .fpl playlists may be filtered before exporting, thus exporting only the Auto-playlists[35].

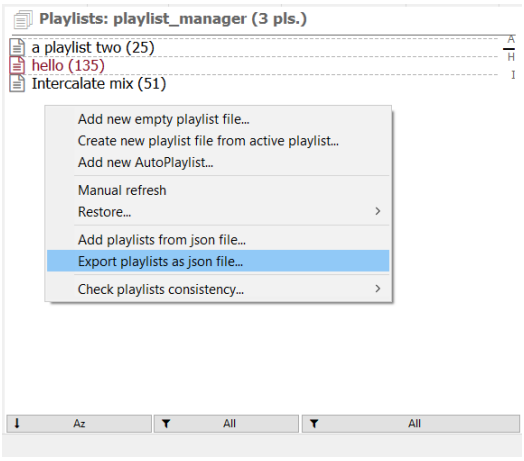


Figure 60: Export all Auto-playlists.

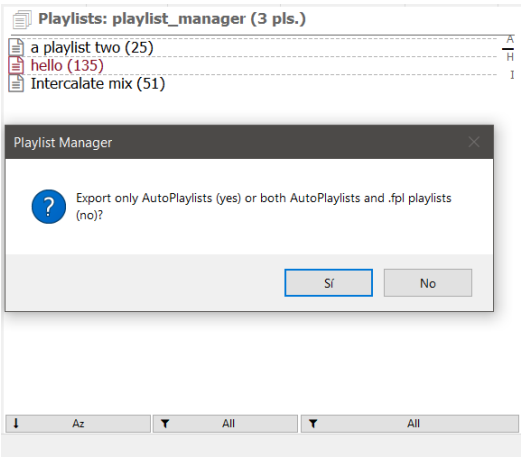


Figure 61: Auto-Playlists and .fpl popup.

[35]Therefore, choosing both playlists types at exporting process is equivalent to simply copying the associated panel json file. Note the advantage is mostly cosmetic for regular users, since .fpl playlists are filtered anyway at importing! It may come handy for advanced users though (for backup purpose or manual .fpl syncing).

13.3 Clone as standard playlist

Auto-playlists may be converted to standard playlists (writable formats [V]) for further editing, sorting or exporting as plain-text files. A new playlist file will be created in the tracked folder with similar name and duplicating the tracks and original sorting from the Auto-playlist.

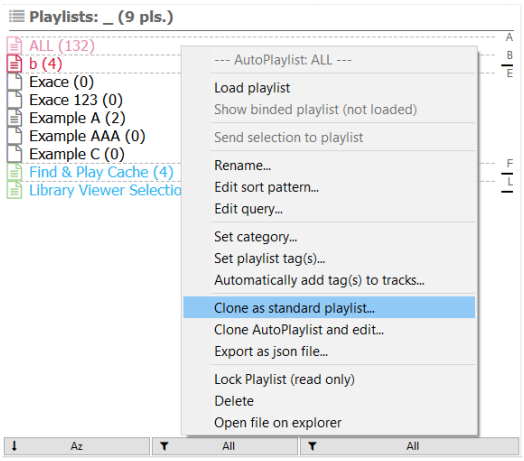


Figure 62: Clone selected Auto-playlist.

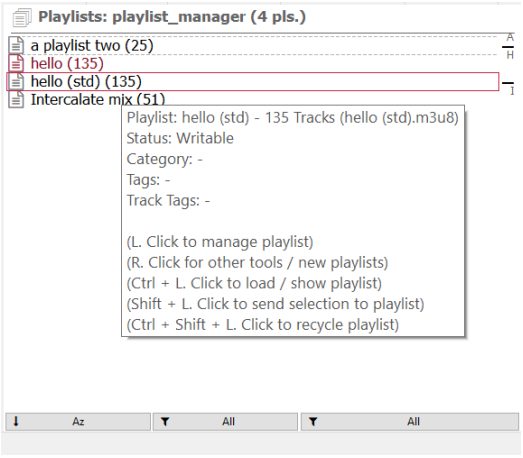


Figure 63: Cloned Auto-playlist as standard playlist.

Additionally, duplicates may be automatically removed according to tag(s) or TF expression(s) by setting 'On AutoPlaylist cloning, filter by....' option. By default is set to 'artist,date,title'[36].

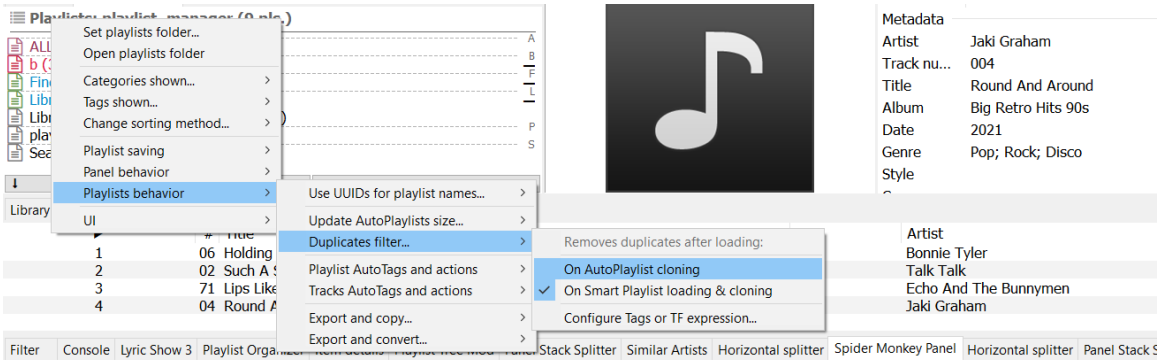


Figure 64: Enabling filtering duplicates for Auto-playlists clones on header menu.

[36] Automatizes the process of removing duplicates by tags after cloning using tools like those found on Playlist-Tools-SMP and automatically fixes one of the worst quirks of Auto-Playlists (having multiple versions of the same tracks)

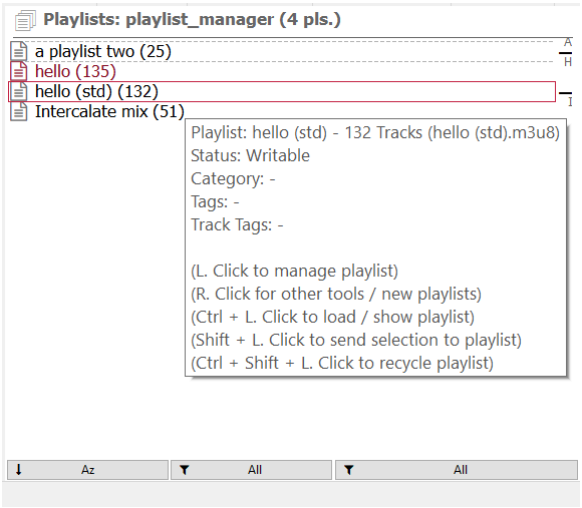


Figure 65: Cloned Auto-playlist now has 3 less tracks (132) than the original (135).

Once an Auto-playlist has been converted, the regular exporting tools may be used [14] to export or convert not only the playlist but also its tracks (for ex. exporting to a portable player). In resume, the procedure to export Auto-playlist tracks and playlists always involves cloning it first as standard playlist[37]. While this may seem like a limitation, it offers multiple advantages over simply converting the tracks using native Foobar2000:

- Duplicates can be filtered automatically. Auto-playlists are non editable, so to achieve the same manually all the tracks would have to be sent to another playlist anyway (and then using additional scripts to filter duplicates by tags).
- The playlist file can be exported along the tracks. Using Auto-playlists only the tracks can be converted/exported, the playlist can not be saved.
- The exported playlist file will point to the tracks using relative paths, contrary to the file written using 'File/Save playlist...'.

13.4 Clone as Auto-playlist and edit query

Alternatively, Auto-playlists may be cloned 'as is' to easily create new Auto-Playlists using an initial query as reference. Name, query and sorting is edited via popups. Note cloning involves copying all tags [27.2], track tags [27.3] and category [27.4] of the original playlist into the new one, which may have further implications (for ex. actions [11] and automatic tagging [12]).

[37]There is an exception, exporting and converting its tracks [14.3] its allowed directly as a shortcut to cloning + exporting later. Conversion to standard playlists is done on the fly without needing intermediate steps. This is offered only for faster processing.

14 Exporting playlists and files

14.1 Copy Playlist file *-format friendly action-*

Exports (a copy of) the selected playlist file to the given path, the final filename may be changed[38]. This is equivalent to open the tracked folder and copying/pasting the file to the desired location.

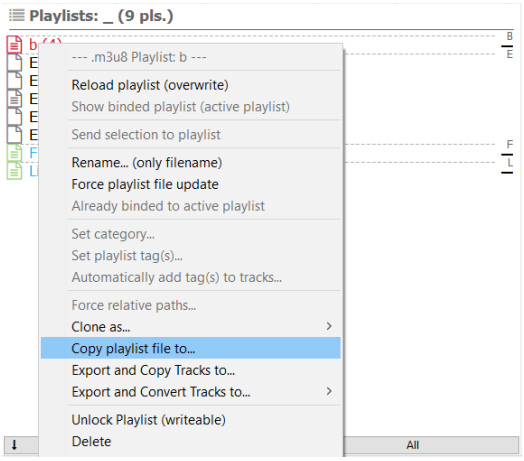


Figure 66: Copy selected playlist file.

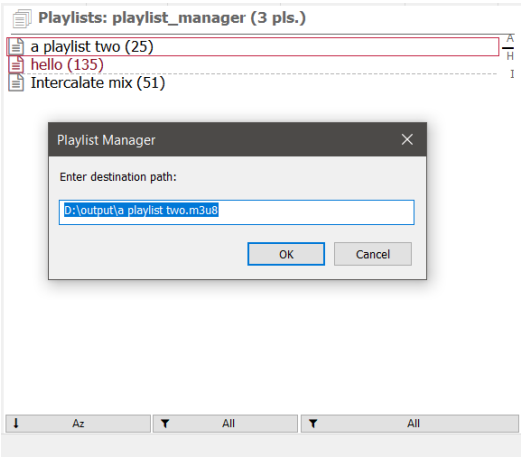


Figure 67: Output popup.

D:\foobar2000\profile\playlist_manager\example.m3u8 → **D:\output\example.m3u8**

This is the only option available for readable-only formats [V] which have a physical file (.fpl). The Auto-playlist counterpart would be exporting as json file [13.2].

[38] If the folder does not exists, it will be created too.

14.2 Export and copy tracks to

Exports (a copy of) the selected playlist file along their tracks to the given path[39], the final playlist filename may be changed.

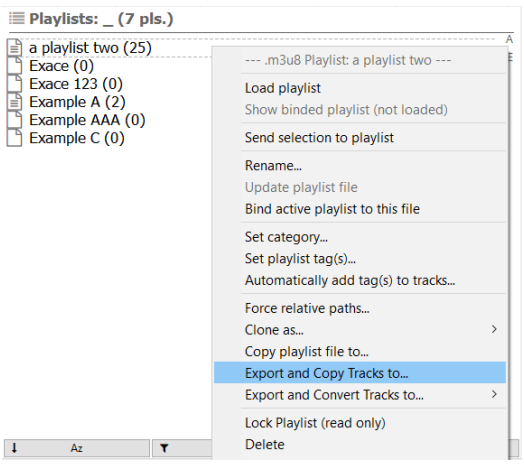


Figure 68: Copy selected playlist file and its tracks.

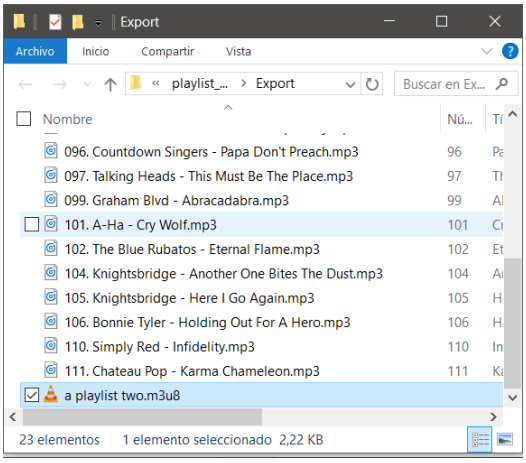


Figure 69: Output folder with all media files along the exported playlist.

Since the tracks are exported 'as is'[40] along the playlist, the exported playlist will be edited to use relative paths, no matter what the original used. This is done to easily load the playlist at any point along its files as a portable solution.

[...]
#EXTINF:259,Jaki Graham - Round And Around
D:\My library\Big Retro Hits 90s\004. Round And Around.mp3
[...]

⇓⇓⇓

[...]
#EXTINF:259,Jaki Graham - Round And Around
.\004. Round And Around.mp3
[...]

This exporting option is not available for readable-only formats [V]. To use it on Auto-playlist, first clone it as standard playlist [13.3], then proceed as usual. To do something similar with .fpl playlists, manually convert them to a writable format and then proceed as usual.

[39] It's obviously recommended to choose a new folder without any content, since it will be filled with all tracks plus the playlist.

[40] No conversion is done, so the file formats will remain the same and they will be perfect copies of the original files.

14.3 Export and convert tracks to

Exports (a copy of) the selected playlist file along their tracks to the given path, the final playlist filename may be changed. The tracks are converted on the process[41] and the exported playlist is edited to use relative paths [14.2]. Note '.\' on filenames ('.\01 - Talikoba.mp3') can be configured on presets to be added or not, thus allowing to prepend a forced absolute path if desired too.

Since the files are converted, instead of being copied, they are ready to use not only on other PCs of Foobar2000 instances but also on portable players, phones, etc. i.e. it may be used as a one way sync tool integrated within the manager and working directly on playlists.

The feature allows to save predefined sets of converter preset + destination folder on the menu for easy access[42]. The entries may be configured, added or removed on the header menu [18].

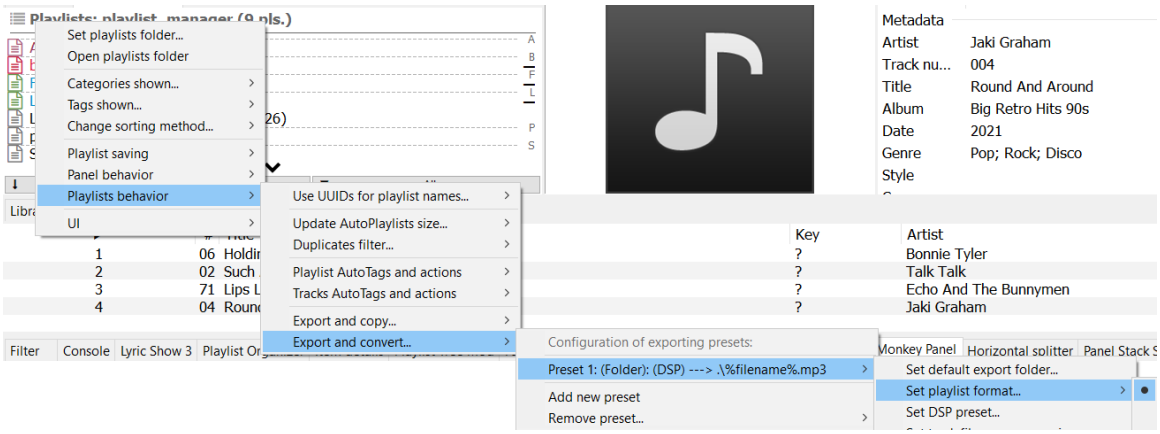


Figure 70: Setting export and convert presets: converter preset, filename mask and output folder.

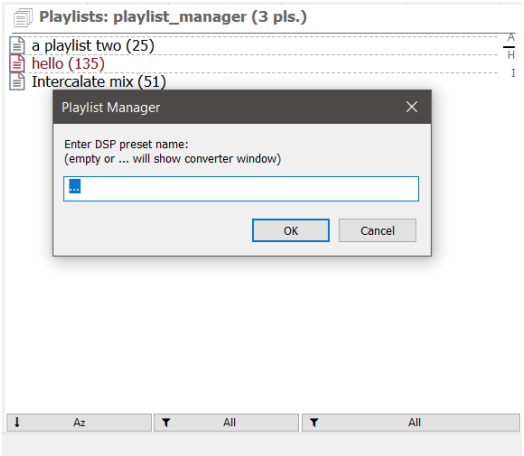


Figure 71: Converter window will be shown at execution.

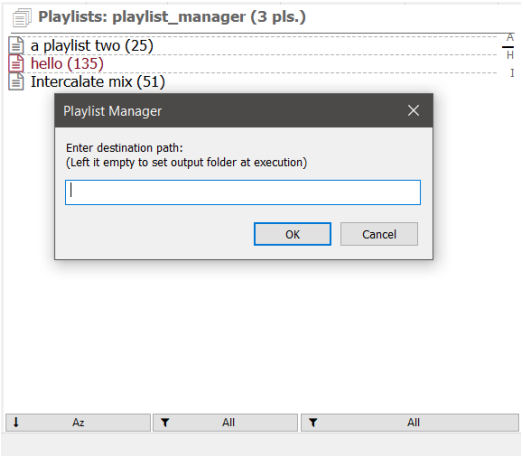


Figure 72: Output path will be asked at execution.

[41] Using pre-defined Converter presets.

[42] For ex. it's possible to have an entry to export playlist to the Ipod and another one for the server, both with different converter configurations and destination folders.

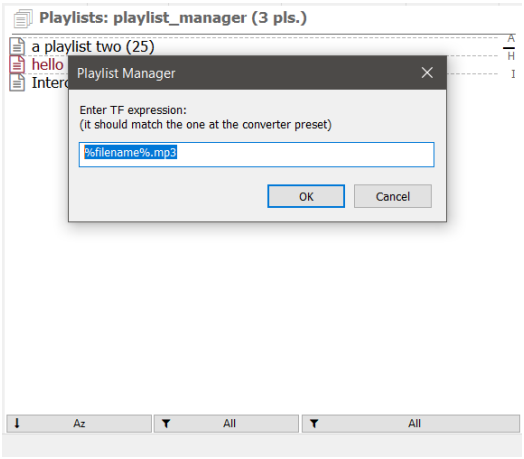


Figure 73: Filename mask must match the one at the converter preset to properly modify names on the output playlist file.

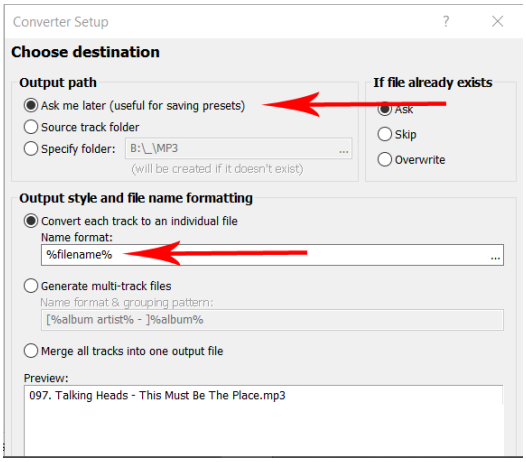


Figure 74: The converter preset can be set to ask for the output path at execution. Note filemask matches the previous one.

On the selected playlist contextual menu, every entry shows the destination folder[43], the converter preset name and the filename mask[44]. When the folder or the preset is not set, '(Folder)' and/or '(DSP)' is shown instead (and they will have to be manually set at execution).

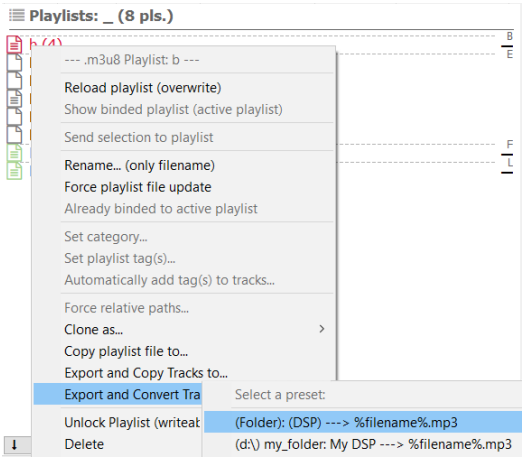


Figure 75: The current preset has a converted preset defined and the filemask, but output folder will be asked at execution.

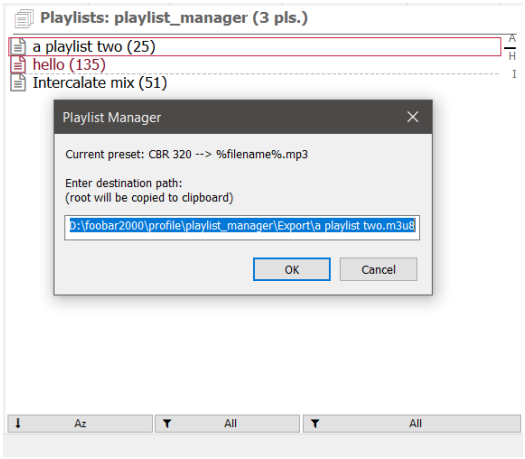


Figure 76: Setting output path. Converter preset should be set to also ask output path before conversion to reuse it in both windows.

This exporting option may not be available for some readable-only formats[V]. It is allowed for Auto-playlists and Smart Playlists, but .fpl playlists must be manually converted or cloned to a writable format, then proceed as usual on the new playlist file.

[43] Along the disk letter in parenthesis.
[44] Must match the one found at the converter preset to work as intended.

15 ListenBrainz integration

The manager allows Playlist ListenBrainz integration (not related to listens), by being able to sync or import playlists found on the server, usually the user profile.

Playlist importing works similar to .xspf format [32], i.e. tracks on server only have a Title, Artist and recording MBID... and they will be associated to local files using those tags when possible. When importing to a plain text format, content resolution will be applied before writing the playlist and non matched items will be discarded. If that behavior is not desired, import to .xspf format to maintain all tracks; matched tracks will be associated to a path and non-matched tracks saved with Title, Artist and MBID tags.

15.1 Sync playlists

Playlists may be exported to or imported from ListenBrainz server. To uniquely identify a playlist, a MBID is used, which is saved on the playlist file. There are multiple scenarios for exporting:

- The playlist, with a MBID, is present on server: exporting will sync the changes. Playlist identification is done by MBID, not name. i.e. multiple playlists with same name may be present on the server.
- The playlist, with a MBID, is not present on the server: A new playlist is created on the server with the playlist name. A new MBID is created and assigned to the playlist file. The old MBID is ignored and overwritten. An error popup will be shown.
- The playlist file does not have a MBID: A new playlist is created on the server with the playlist name. A new MBID is created and assigned to the playlist file.

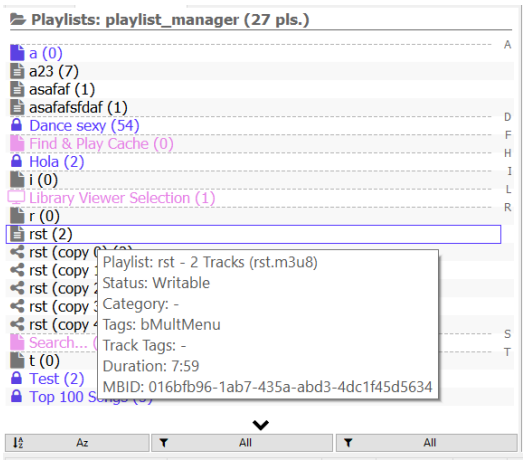


Figure 77: Playlist MBID shown on tooltip.

Importing will retrieve the associated playlist by MBID from the server and overwrite the local file with it. This option is only available when the playlist file has a MBID. In case it's not found (or not valid) an error popup will be shown and no changes applied.

15.2 Import playlist from server

A playlist may also be imported directly from the server, without a previous file, by giving a MBID. In this case the playlist file does not need to be associated to the user account, i.e. a playlist may be imported from any account as long as it's public.

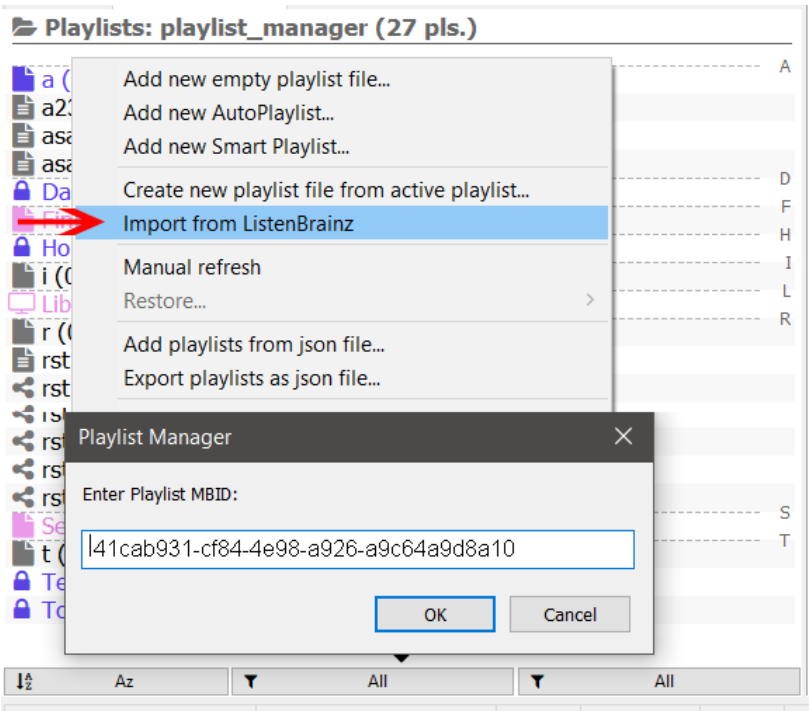


Figure 78: Import by playlist MBID.

15.3 ListenBrainz user token

An user token is required for all ListenBrainz functionality; it may be set on the header menu [18]. Token encryption is allowed with a password, which will be asked on every related action afterwards.

16 Additional tools

16.1 On selected playlist

The following tools can be found on the selected playlist contextual menu [18].

16.1.1 Force relative paths *-format friendly action-*

Paths within a playlist can be converted to relative paths stripping all but the filenames[45]. It's a destructive action, which edits the playlist file and can not be undone... although a backup can be found at the Recycle Bin as usual [8.8]:

> [...]
> #EXTINF:259,Jaki Graham - Round And Around
> **D:\library\Big Retro Hits 90s\004. Round And Around.mp3**
> [...]
>
> ⇓⇓⇓
>
> [...]
> #EXTINF:259,Jaki Graham - Round And Around
> **.\004. Round And Around.mp3**
> [...]

[45] It's easy to see this is equivalent to using the 'export and copy tracks' feature [14.2] without copying the tracks and overwriting the original playlist file

16.1.2 Send selection to playlist *-format friendly action-*

Menu entry and shortcut [18] are functionality-wise the same. Sends currently selected tracks to the selected playlist without opening or switching to it: is instant and tracks may be easily "drag and dropped" without changing the active playlist.

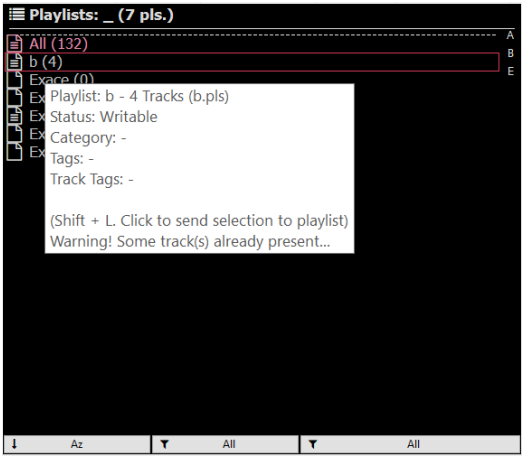


Figure 79: Send selection shortcut.

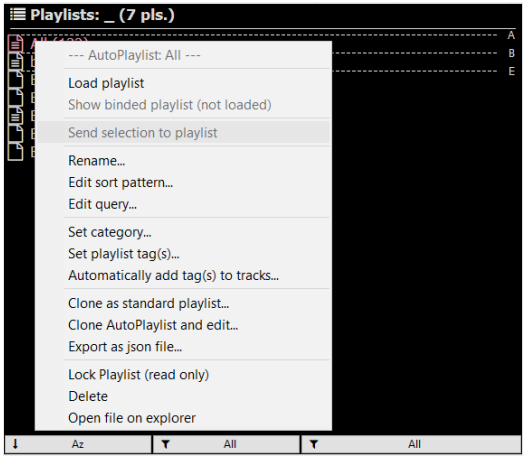


Figure 80: Menu entry.

As a bonus, duplicates are checked by default (may be disabled at configuration), so adding multiple times the same tracks is not allowed; duplicates are simply skipped whenever adding multiple tracks. The tooltip also warns when trying to add duplicates, so it may be used to easily check if a track can be found within a playlist [22.8].

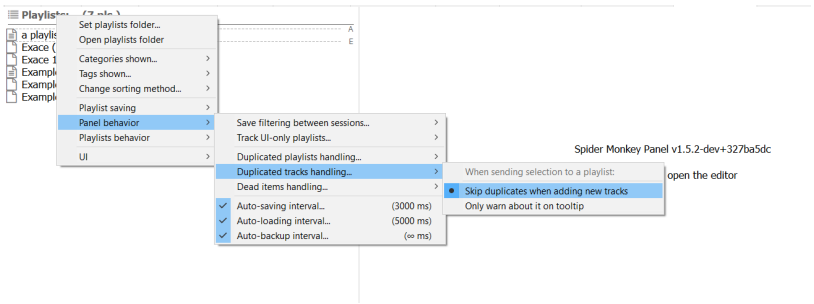


Figure 81: Skip duplicates configuration.

The main advantage of this action compared to opening the playlist into the UI and then adding the tracks copying/pasting is not only faster processing and ease of use. (Auto-)saving a playlists involves changing its format to the default one set on the panel [8.4], while directly sending tracks to a playlist will simply append them to the playlist without changing its original format. Therefore you can maintain playlists in multiple formats [V] as long as a save action is not performed (manually or on auto-saving).

16.1.3 Clone as

Playlists may be cloned as any of the writable formats [V], creating another file into the tracked folder. New file is automatically renamed with ' (copy X)' at the end. Cloning may also be used to create a copy of any playlist as UI-only playlist[36]. This is meant to load a playlist into the UI without binding it to a physical file or affecting the original one[46].

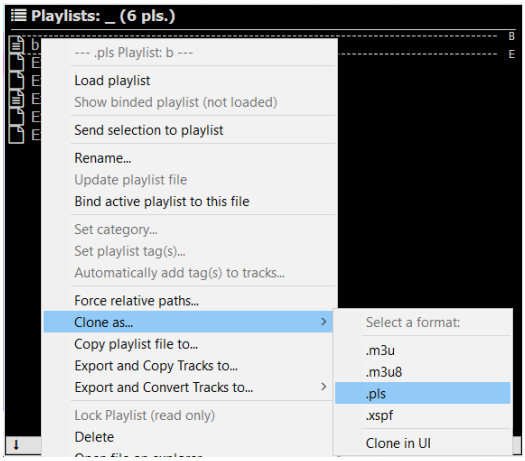


Figure 82: Clone selected playlist.

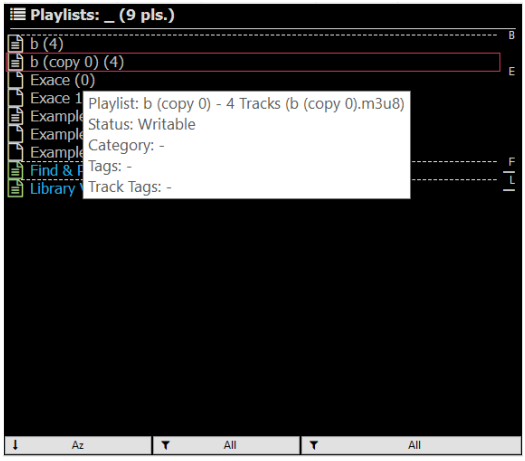


Figure 83: Cloned playlist with new name.

Beware of metadata losses when cloning to a playlist format without advanced metadata support (ie. to .pls [30]), not all formats may allow storing all the metadata from the original source [28].

Playlists may also be cloned loading them and creating a new one from the active playlist (the loaded playlist), as explained in the FAQ [VIII]. This method discards any original metadata since it simply creates a new playlist with the given tracks.

Auto-playlists have similar features to clone them as standard playlists [13.3] (using the output tracks) and as Auto-playlists [13.4] (reusing the query).

[46]It would be "equivalent" to previously locking a playlist file and then loading it, but this method skips setting the metadata first and ensures no changes are made to the original file in any case. Also, cloning it into the UI does not create another physical playlist file... so this is a "read-only" action.

16.2 On entire list *-format friendly action-*

The following tools can be found on the list contextual menu [18]. Executed asynchronously [VIII].

16.2.1 Mixed absolute and relative paths

The manager can check all the playlists currently tracked to ensure there are no playlist files with both absolute and relative paths at the same time. Such files are probably an error, whether intentional or not, and should be fixed in most cases. For ex:

[...]
#EXTINF:141,Jeffery Mykals - Gyal Bad
..\music\Jeffery Mykals - Gyal Bad.mp3
#EXTINF:271,Jeffery Mykals - Chico & Voicemail
D:\music\Jeffery Mykals - Chico & Voicemail.mp3

The tool will span a popup reporting a list of playlists with such 'problem'. If all items exist (no dead items), as soon as the playlist is rewritten by the manager it will be automatically fixed[47].

16.2.2 Dead items

The manager can check all the playlists currently tracked for dead items on them, whether the tracks are on current Foobar2000's library or not. Dead items are considered files which don't exist at their path, no matter if it's a relative or absolute path.

The tool will span a popup reporting a list of playlists with dead items (but will not list the items their-selves). To find such items or replace them with items from current library use a dedicated tool like the one found at Playlist-Tools-SMP[48].

16.2.3 External items

The manager can check all the playlists currently tracked for external items on them, i.e. tracks not present on Foobar2000's library but which exists at their path. Note external items are technically not dead items, since they do exist outside Foobar2000 database.

[47] It will use the current path configuration, thus converting all paths to absolute or relative paths, after loading the playlist within Foobar2000, on auto-update or via manual update on selected playlist menu [18].
[48] Look for 'Playlist Revive'.

16.2.4 Duplicated items

The manager can check all the playlists currently tracked for duplicated items on them, i.e. two tracks with the same path. Note there is a limit though, if absolute and relative paths are intentionally mixed and 2 paths point to the same physical file, they will not be considered duplicated. For ex[49]:

[...]
#EXTINF:141,Jeffery Mykals - Gyal Bad
..\music\Jeffery Mykals - Gyal Bad.mp3 ∗
#EXTINF:141,Jeffery Mykals - Gyal Bad
D:\music\Jeffery Mykals - Gyal Bad.mp3
[...]
#EXTINF:141,Jeffery Mykals - Gyal Bad
..\music\Jeffery Mykals - Gyal Bad.mp3 ∗

The tool will span a popup reporting a list of playlists with duplicated items (but will not list the items their-selves). To find such items or remove them use a dedicated tool like the one found at Playlist-Tools-SMP[50]

16.2.5 Blank lines

The manager can check all the playlists currently tracked for blank lines on them (at any position). Note some playlist formats are supposed to only store 1 line per file (.strm) or may allow blank lines which are simply skipped by most players (M3U).

The tool will span a popup reporting a list of playlists which contain blank lines (this includes empty lines and lines with only blank spaces).

[49] Only the tracks with an ∗will be considered duplicates.
[50] Look for 'Duplicates and tag filtering'. Foobar2000's main menu 'Edit/Remove duplicates', after loading the playlist, can also be used.

17 Manual refresh

Since auto-loading may be disabled [10.2] or the refresh time set too high, there is an option on the list contextual menu [18] to force updating the tracked folder and its playlists. Any new file found will be added to the list.

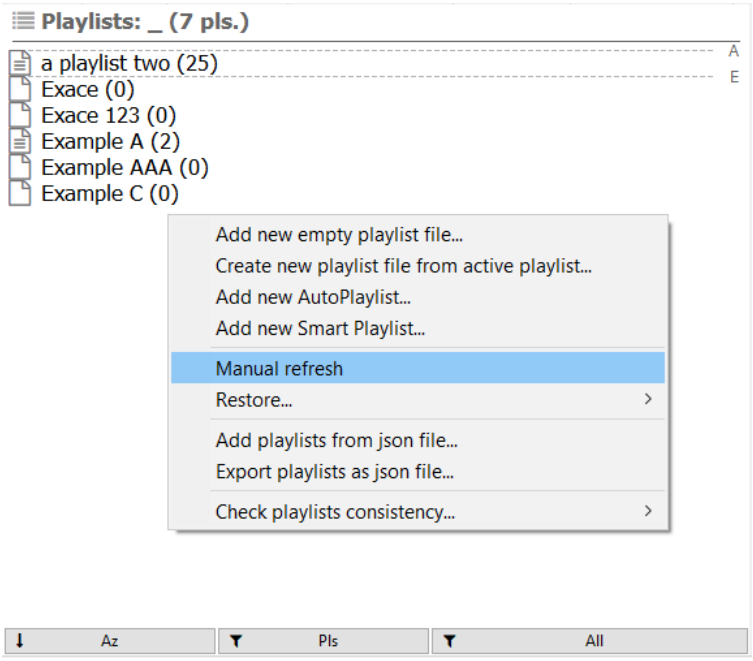


Figure 84: Force manual refresh of list.

As a side effect, Auto-playlists's metadata will be refreshed (size). Note this is the only way to do it unless configured to do so automatically [23.2] at startup or individually loading them. Since doing it at startup may involve a small increase in loading time [VIII], it may be better to refresh them manually from time to time using this option on resource constrained PCs,

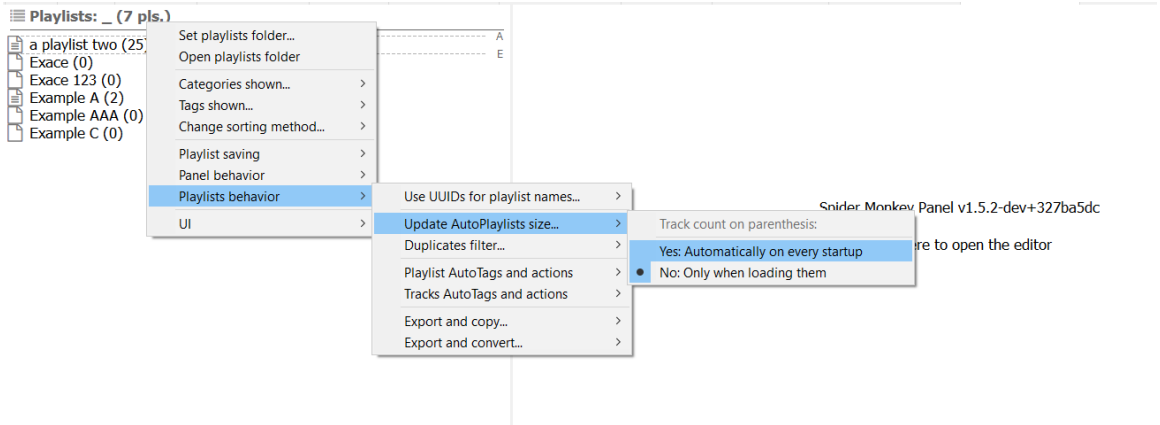


Figure 85: Automatically update Auto-playlists size on startup.

18 Shortcuts

Note selected playlist related actions are always assigned to left click, while the rest are usually assigned to right click (except category cycling). All features can be reached using the following mouse and keyboard shortcuts:

- **Open menus (*clicking on item within parentheses*)**:
  + **Left Click (*playlist*)**: Open selected playlist contextual menu.
  + **Right Click (*list*)**: Open list menu; new playlist and other playlist tools.
  + **Right Click (*header*)**: Open header menu. General manager configuration.
  + **Right Click (*buttons*)**: Open buttons menu. Sorting and filtering.
- **Playlist actions (*clicking on playlists -user configurable shortcuts [18]-*)**:
  + **Double Click**: Load selected playlist / Make bound playlist active.
  + **Ctrl + Left Click**: Load selected playlist / Make bound playlist active.
  + **Shift + Left Click**: Send current selection to selected playlist.
  + **Ctrl + Shift + Left Click**: Clone playlist in UI.
  + **Middle Click**: Multiple playlist selection.
- **Current view actions (*clicking on header*)**:
  + **Double Click**: Category cycling.
  + **Single Left Click**: Select active playlist or playing playlist in the manager (if possible).
  + **Single Middle Click**: Select all playlists.

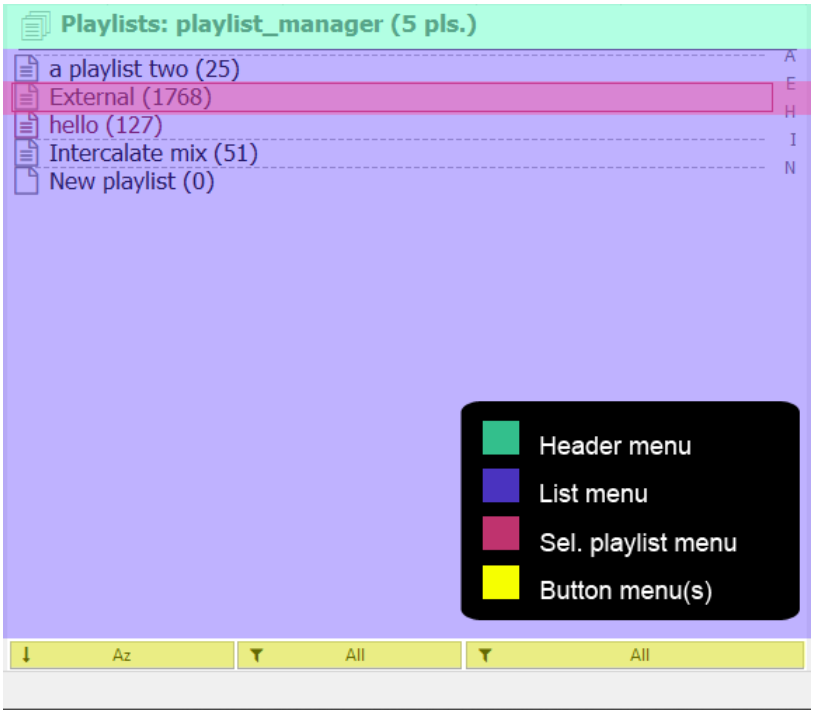


Figure 86: Opened menus change according to mouse position while clicking.

19 Dynamic menus

Some **playlist and panel actions** can exposed as main menu entries which can be found at '*File\SpiderMonkeyPanel\ScriptCommands\PlaylistManager*'. As a menu entry, **they can be run via CMD or assigned to a keyboard shortcut**, used by online controllers [VII], etc. To use this feature, 'Dynamic menus' must be enabled on the header menu [18]:

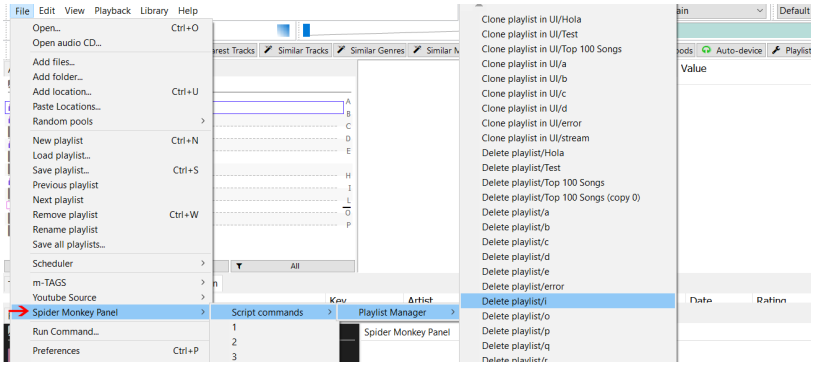


Figure 87: Menu at '*File\SpiderMonkeyPanel\ScriptCommands*'

Additionally, to run those menus entries via CMD or within an online controller [VII] there are 2 more requisites:

- **foo_run_main:** to run dynamically created main menus with other plugins, CMD, ...
- **Unique panel names:** every SMP panel which is meant to be used with dynamic menus must have a unique name. As shown on the previous figure, the menu is created using the Panel Name as reference... so having multiple panels with the same name will provoke collisions. This limitation is only for the name, not the script, i.e. you may load the same script in multiple panels as long as the panels are named different.

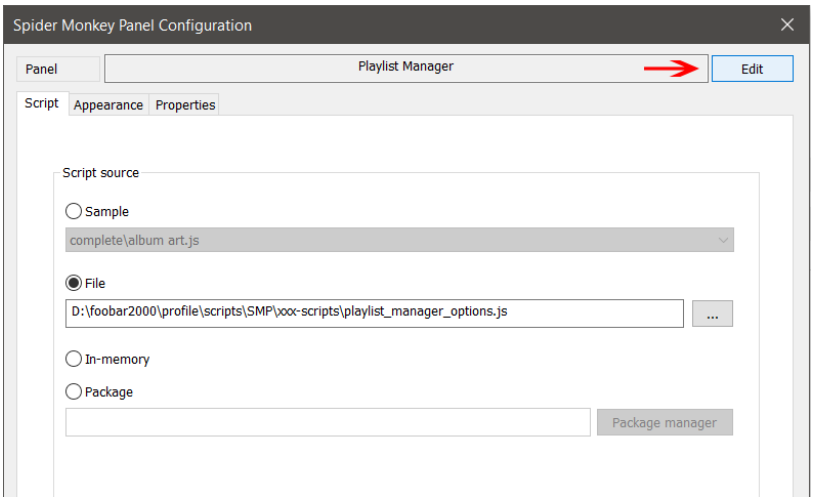


Figure 88: When having multiple playlist manager panels, names must be edited.

19.1 Available actions

Each available action along its description (see references) can be found below:

- Playlist context: one menu entry per playlist (associated by name).
  - + Clone in UI [16.1.3].
  - + Delete [8.8].
  - + Load [8.6].
  - + Lock/Unlock [8.9].
  - + Send selection to [16.1.2].
- Playlist group context: one menu entry per action (applied to all playlist with 'bMultMenu' tag [11]).
  - + Load [8.6].
  - + Clone in UI [16.1.3].
- Panel context:
  - + New empty playlist [8.5].
  - + New playlist (active) [8.5].
  - + Manual refresh [17].

Note playlist context actions are dependent on playlist names, i.e. whenever a name changes the associated menu entry changes too. Beware of this when using them on CMD or Keyboard shortcuts.

19.2 Keyboard shortcuts

Once "Dynamic menus" are enabled [19], entries can be directly assigned to a keyboard shortcut the same than any native foobar2000 menu entry.

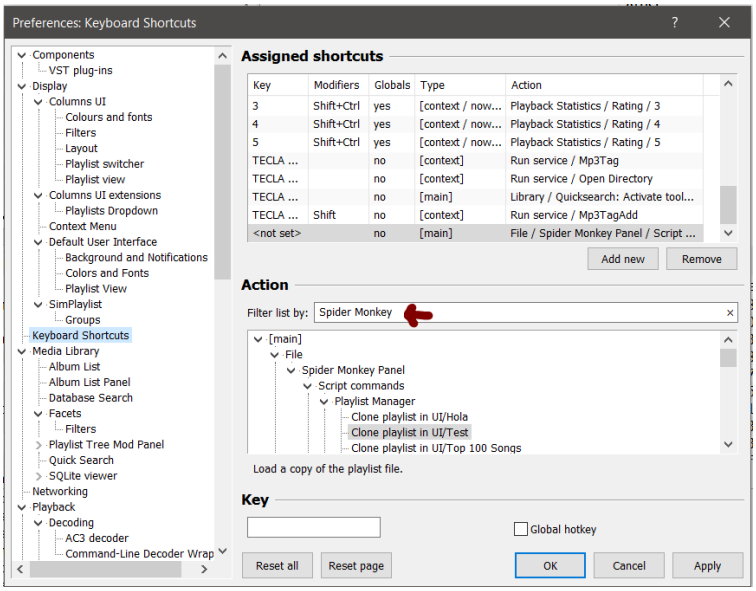


Figure 89: Setting a keyboard shortcut to SMP menu entry.

20 Multiple selection

By using the multiple selection shortcut [18], multiple playlists may be selected at the same time. In such case, the 'selected playlist menu' [18] applies to the entire selection of playlists instead of a single one.

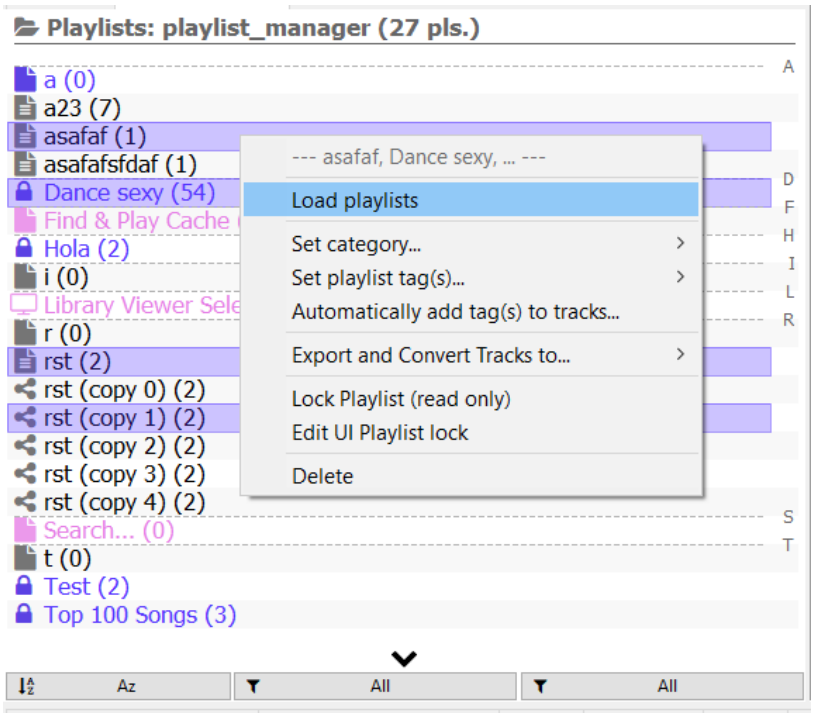


Figure 90: Multiple selection menu.

The set of available entries differs to the single playlist menu, and any action is applied partially to allowed playlists. For ex: Using the 'Lock playlist (read only)' menu entry on a list of playlists will only lock those playlists that are not already locked. Similar behavior applies when setting a tag or category.

- Load playlists [8.6].
- Clone in UI [16.1.3].
- Set category [9].
- Set playlist tag(s) [9].
- Automatically add tag(s) to tracks [12].
- Export and Convert tracks to [14.3].
- Lock/Unlock playlists [8.9].
- Edit UI playlist lock [8.10].

Part IV

UI

21 Features

- UI re-sizable on the fly. i.e. it will adjust layout to panel size.
- Loaded and Now playing playlist indicators: —/ ◁
- Empty / not empty playlist indicators. To be used as fallback when size is not shown.
- Font Size (configurable).
- Separators between different names/categories (configurable).
- Colors for different playlists types, status, text, background and selection (configurable).
- Different playlist icons per format (configurable).
- Alternate row shading and automatic text color changing for readability.
- Accessibility options and presets (color blindness, grey scale).
- Multiple filtering options (which can be combined).
- Quick-search entire words (letters and numbers) according to sorting mode [24].

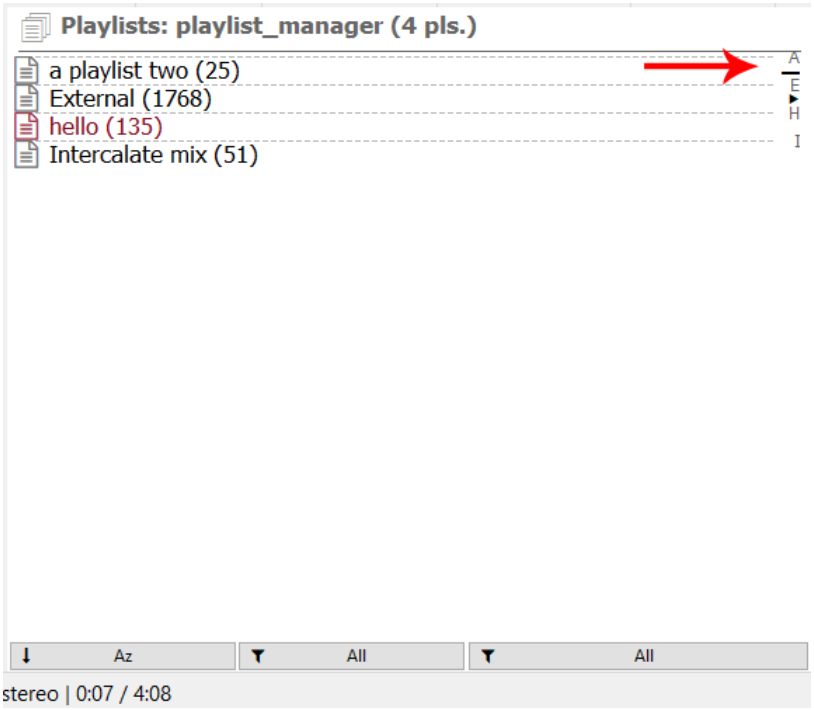


Figure 91: Now playling ('External'), loaded ('a playlist two') and two non loaded playlist.

22 List view

The main panel view features a simple listing of all currently tracked playlists (whether they are physical or virtual -json- files). This view can be filtered according to these parameters:

- By type of playlist: Auto-playlists & Smart playlists, standard playlists or all.
- By lock status.
- By extension.
- By chosen categories (virtual folders).
- By chosen tags.

The current view settings are always maintained on subsequent foobar startups unless otherwise configured. Note tag filtering is always reset since it's meant for informative purposes only[51], not for playlist categorization.

Name / category separators may be shown on the UI to easily identify where the next char begins (123, A, B, ...). This features depends on the sorting mode being used.

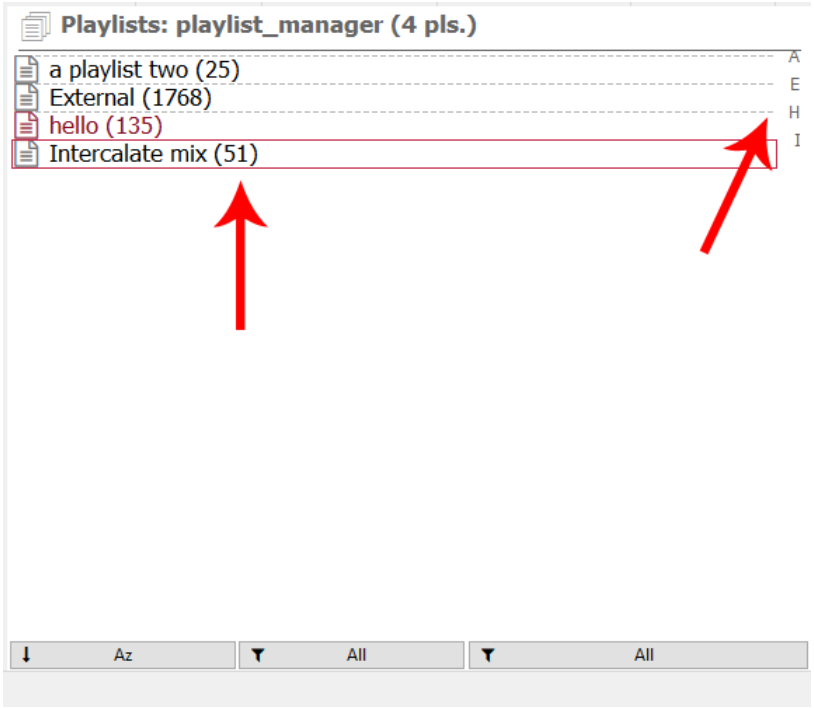


Figure 92: 'A', 'H' and 'I' separators are shown for names.

[51] In fact is also used for some complex playlist automatic actions.

22.1 Sorting *-permanent-*

List view may be sorted by name, category, size, tag (first one) or last modified date. By default playlists are sorted by name, using natural sorting (Az). Ordering may be changed using the appropriate buttons. The selected sorting mode is maintained on subsequent startups.

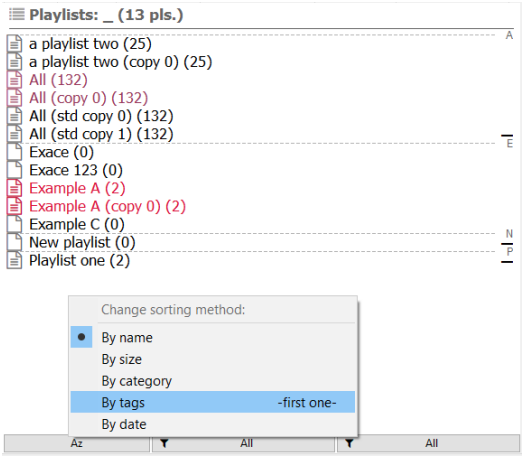


Figure 93: Sorting with the different modes.

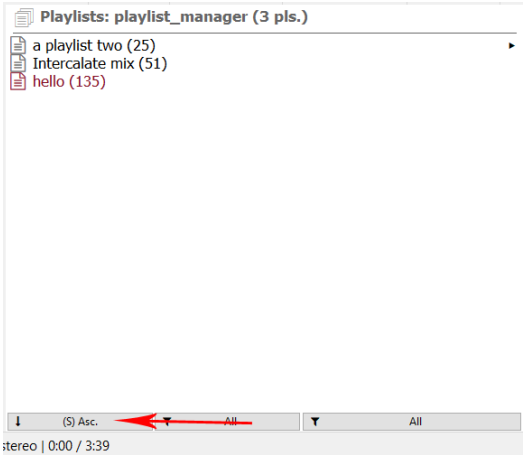


Figure 94: Sorted by size (note the '(S)').

22.2 Lock status, extension, playlist type filtering *-permanent-*

Filter buttons may also be used to filter by other metadata or playlist format. The procedure is similar to category or tag filtering. Like category filtering, the selected filters are maintained on subsequent startups (along the buttons' configuration).

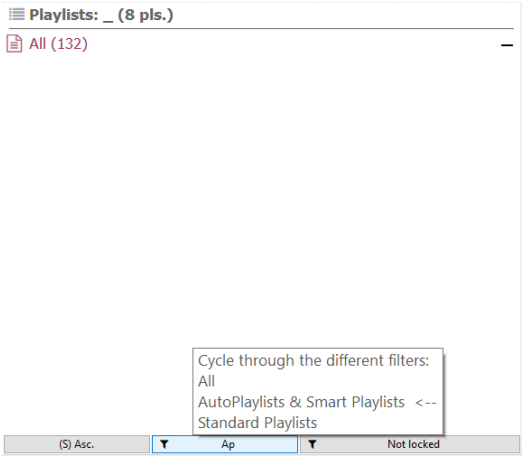


Figure 95: Playlist type and lock status.

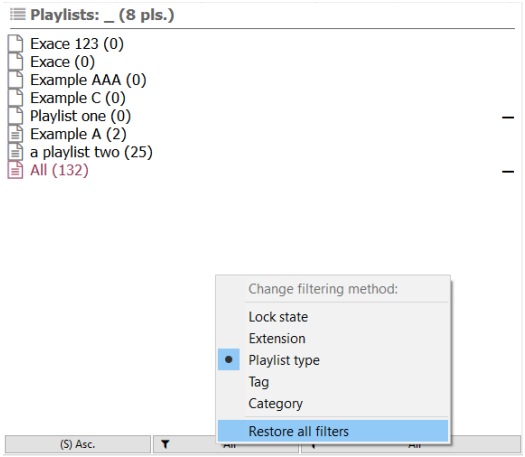


Figure 96: Restoring resets all filters.

22.3 Selective category filtering *-permanent-*

Playlists may be filtered by category (like virtual folders) and multiple individual selections are allowed via menu (for ex. to display all but one specific category). When lists are being filtered by category, an indicator is shown in the header text. The configured category filtering is maintained on subsequent startups.

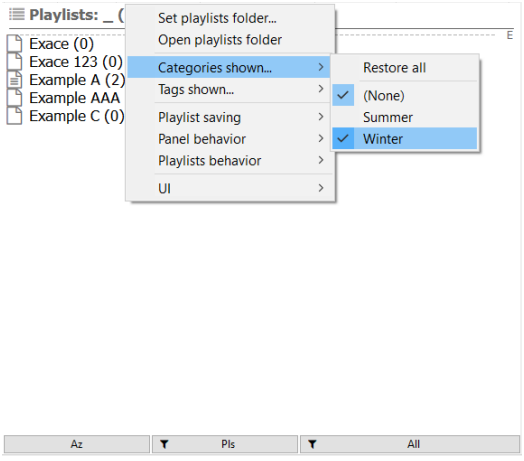


Figure 97: 'Summer' category has been excluded, allowing none and 'Winter'.

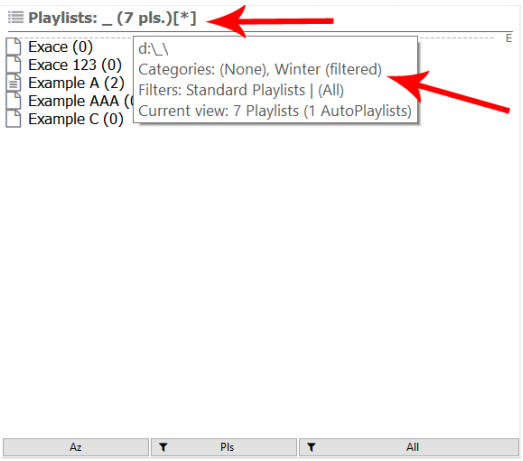


Figure 98: The header and tooltip indicates a filter is active.

An additional way to filter by category is cycling the current category being shown (one by one) [18]. This works in conjunction with playlists being allowed to only have one category at the same time, so no playlist will be shown twice while cycling. Following the same principle, category inheritance on playlist creation [9.1] is not available when showing multiple categories at the same time. Playlists may only have one category at most, so to make use of metadata inheritance only one category must be enabled on the filter.

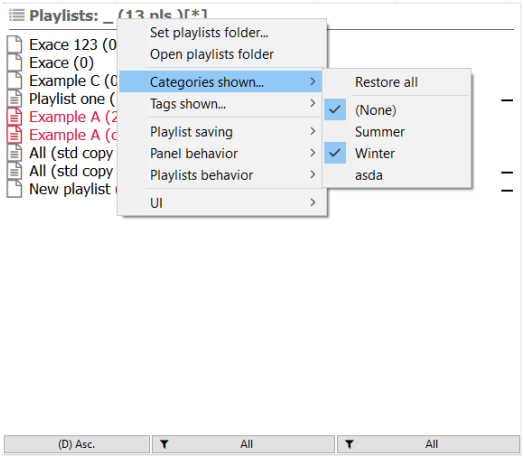


Figure 99: Since 2 categories are being shown, category inheritance is disabled.

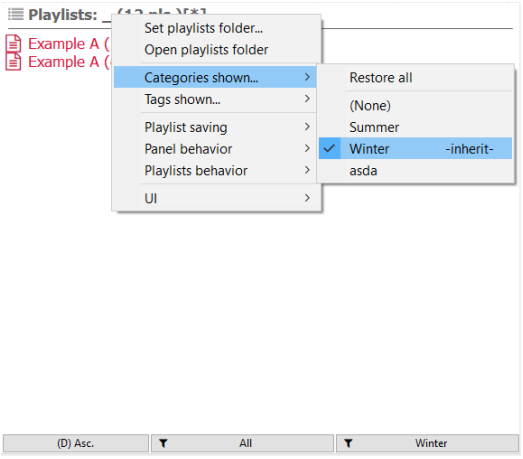


Figure 100: New playlists will have 'Winter' category by default.

22.4 Category filtering cycling *-permanent-*

Selective category filtering [22.3] may be used to exclude or show multiple categories, but fast cycling -one by one- is also allowed using any of the two configurable filter buttons.

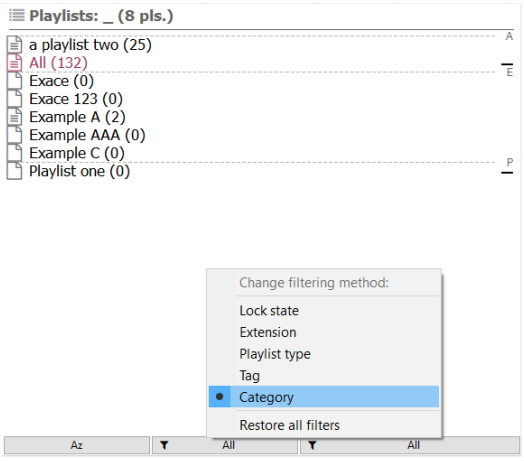


Figure 101: Filter button must be set to category.

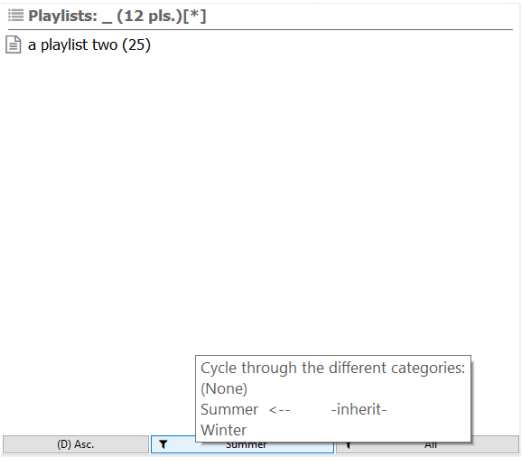


Figure 102: Clicking on it cycles through all categories.

Alternatively, double clicking on the header [18] will perform the same action than the aforementioned button; both can be used at the same time and the UI will be synced accordingly.

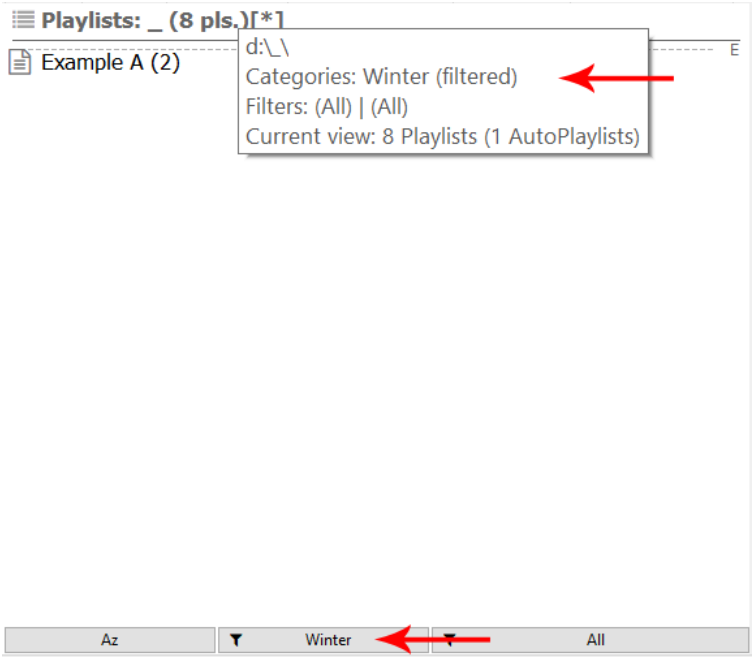


Figure 103: Double clicking on the header also cycles to the next category ('Winter').

22.5 Selective tag filtering *-temporal-*

Playlists may also be filtered temporarily by tags; it gets reset on subsequent startups. Therefore tags should only be used for informative purposes but not for categorization. Note a playlist may have multiple tags at the same time.

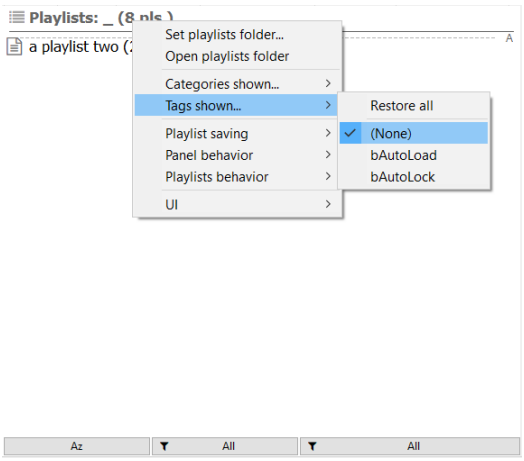


Figure 104: 2 tags have been exluded, showing only playlists without tags.

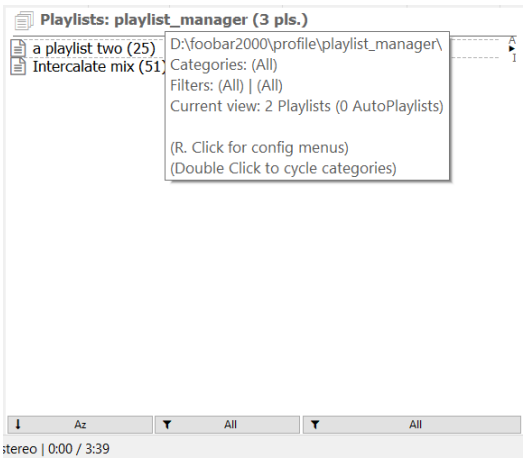


Figure 105: Note the header does not warn about filtering for tags.

22.6 Tag filtering cycling *-temporal-*

Similar to categories, tags may be cycled using the filtering buttons. There is no additional mouse shortcut for it though and tag filtering is always reset on panel reloading.

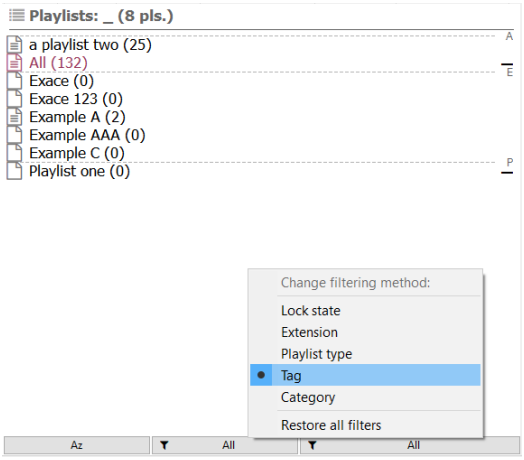


Figure 106: Filter button must be set to tag.

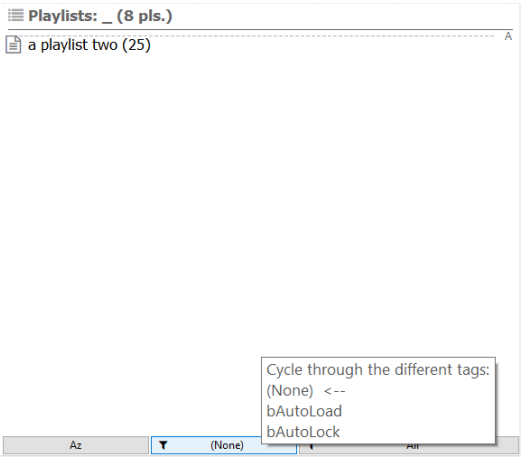


Figure 107: Clicking on it cycles through all tags.

22.7 Reset filtering

Category and tag filtering can be reset at any time using the header menu [18] and clicking on 'Restore all' at the appropriate submenu.

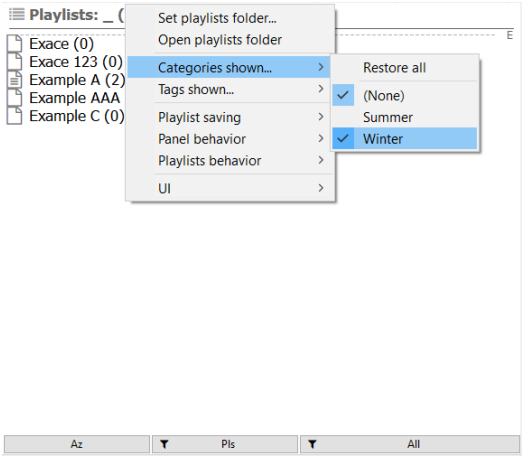


Figure 108: Reset all categories.

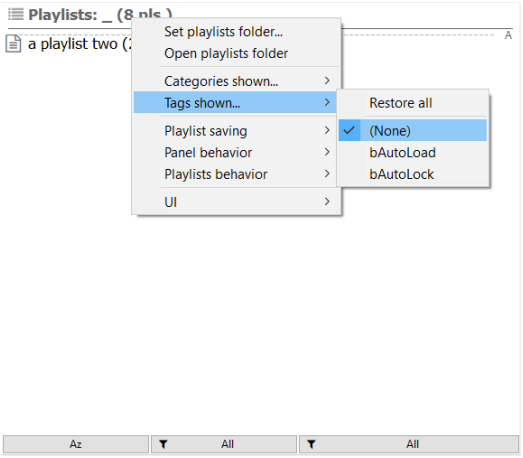


Figure 109: Reset all tags.

Filters may also be reset globally to show all playlists, this also applies to categories and tags (no matter if they were filtered using the buttons or the header menu).

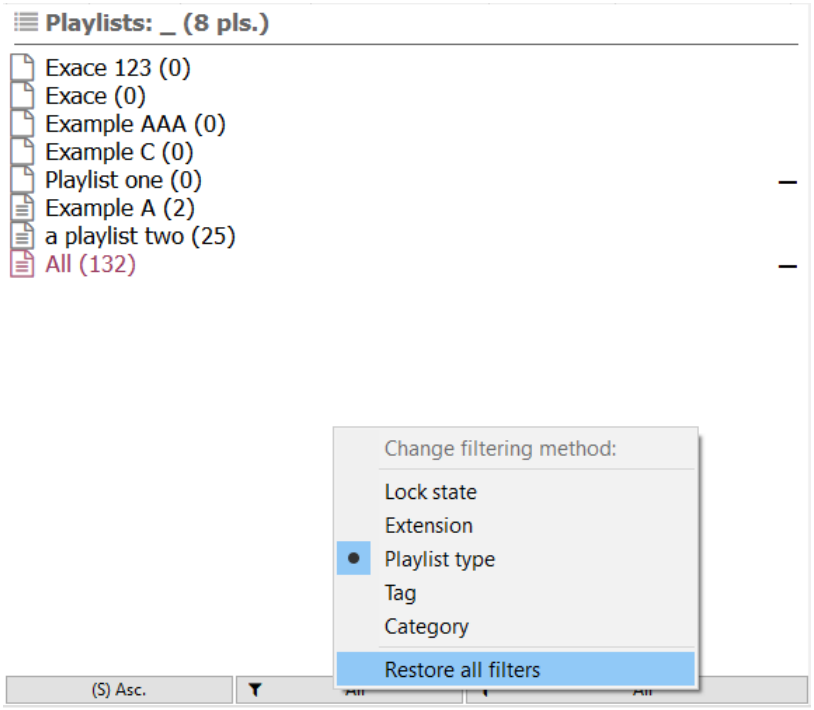


Figure 110: Reset all existing filters.

22.8 Tooltip

Tooltips show different info according to the mouse position:

- Header:
  * Absolute path of currently tracked playlist folder.
  * Filters used on current view.
  * Categories shown.
  * Playlist and Auto-playlists shown on current view (filtering included).
  * Shortcuts info (configurable).

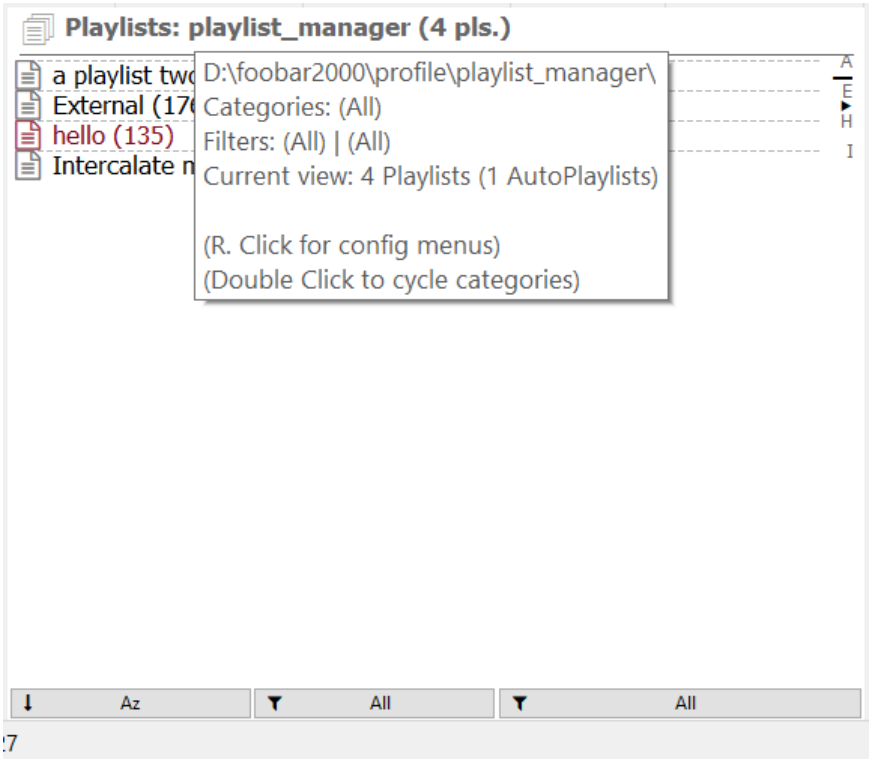


Figure 111: Header tooltip with general info about the current panel, all playlists, filters applied, etc.

- Playlists: (related to selected playlist)
    * Playlist type: Playlist / Auto-Playlist
    * Name plus UUID.
    * Playlist size (tracks). Configurable for Auto-playlists (output by query).
    * File name with extension.
    * Status (lock).
    * Category / Tag(s).
    * Track Tag(s).
    * Duration.
    * Playlist MBID [15].
    * Query. Sort Pattern. Limit. (only Auto-playlists / Smart playlists)
    * Shortcuts info (configurable).
    * Warns if selected tracks are already present on playlist.

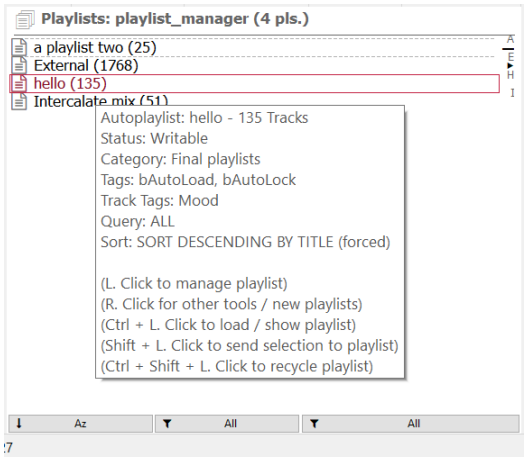


Figure 112: Playlist tooltip.

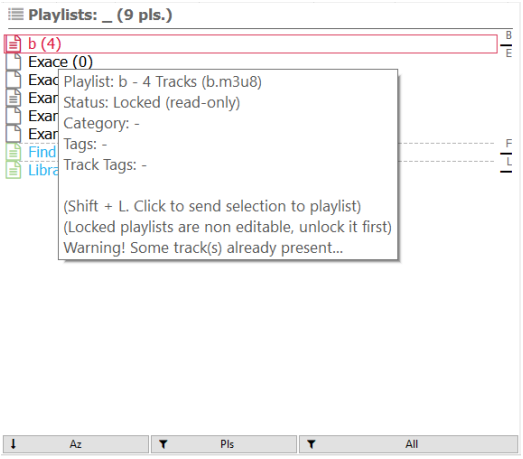


Figure 113: Duplicates warning on playlist tooltip.

23 Customization

23.1 Color customization

- Background: panel background.
- Buttons text: for text within buttons, in case you change the SO theme.
- Standard text: for standard playlists and header. By default inverse of background.
- Special playlists: different color for Auto-playlist, Smart playlists [35] and Locked playlist.
- UI playlists: different color for non editable (UI-only) playlists.
- Current selection: currently selected playlist rectangle.

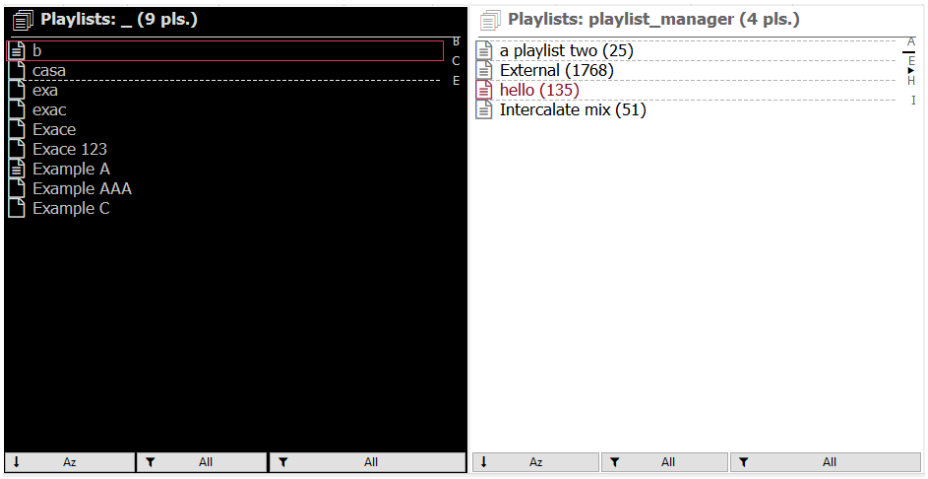


Figure 114: 2 panel instances with different UI colors set.

23.2 Other configuration

- Tooltip: Shortcuts info [18] can be shown or hidden.

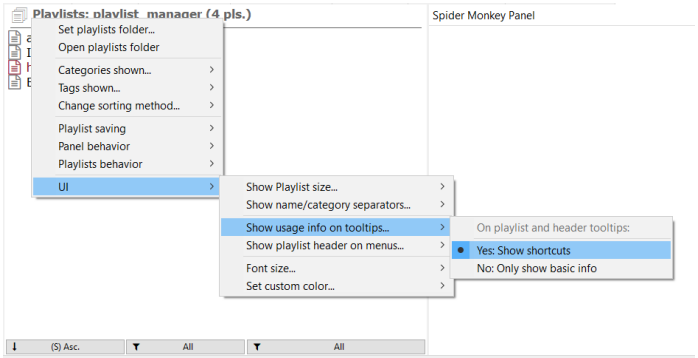


Figure 115: Header menu to enable shortcuts info.

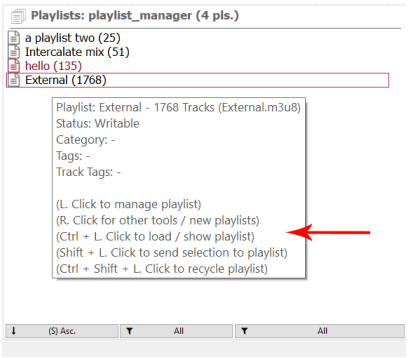


Figure 116: Tooltip.

- Menus: menu header for selected playlist contextual menu [18] can be shown or hidden.

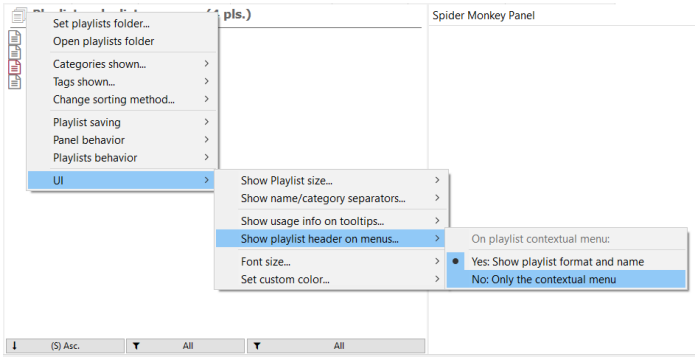


Figure 117: Header menu to enable playlist headers.

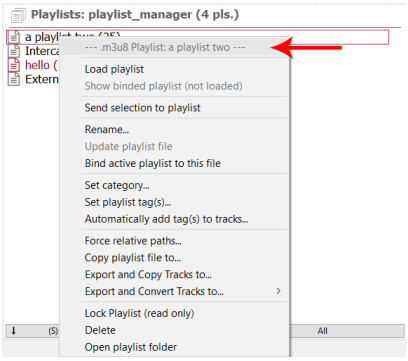


Figure 118: Contextual menu.

- Separators: Name or Category separators may be shown when sorting by those values.
- Font size: applies to all text within the panel.
- Playlist size: Track count may be shown on parenthesis along playlist names. An additional configuration allows to refresh Auto-playlists on startup.

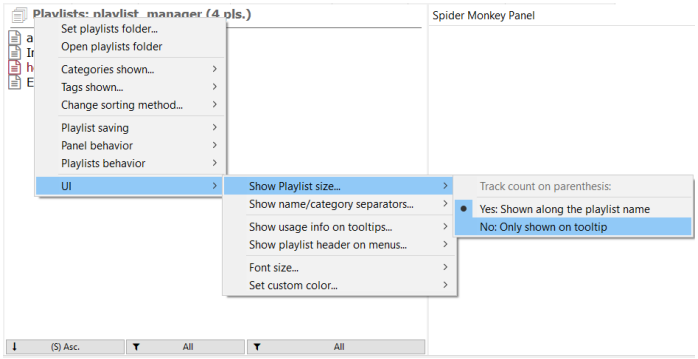


Figure 119: Show size on list.

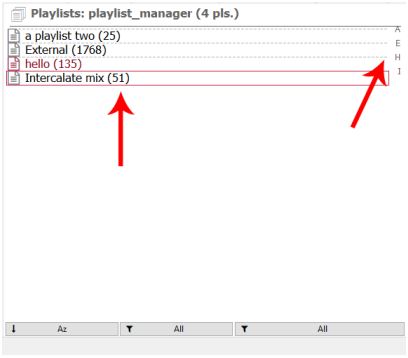


Figure 120: Size and separators are shown for playlists.

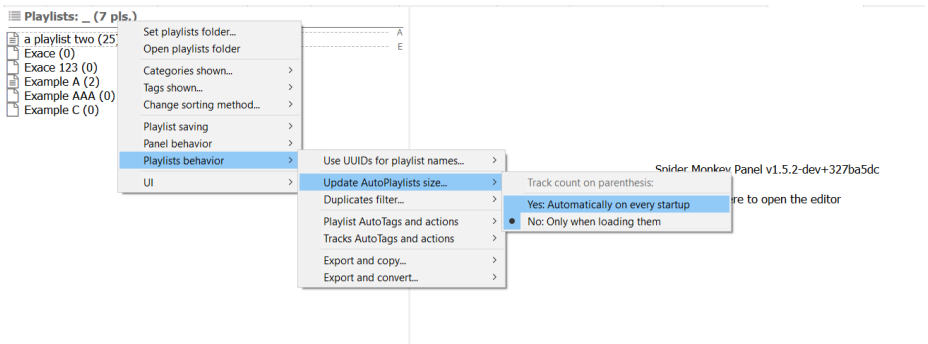


Figure 121: Automatically update Auto-playlists size on startup.

23.3 Accessibility

Color customization may be easily swapped using presets. Predefined presets for color blindness (deuteranopia), grey scale, default and dark theme may be found on the menus:

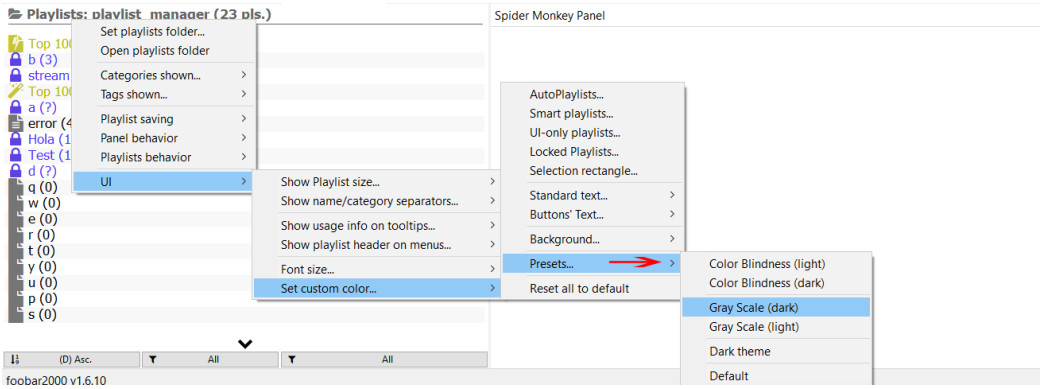


Figure 122: Color presets to change the entire UI.

Additionally, all playlist have an associated unique icon according to their format [V]; the same applies to locked and empty playlists. Therefore it should be possible to identify the playlist format / state without the color codes or the tooltip.

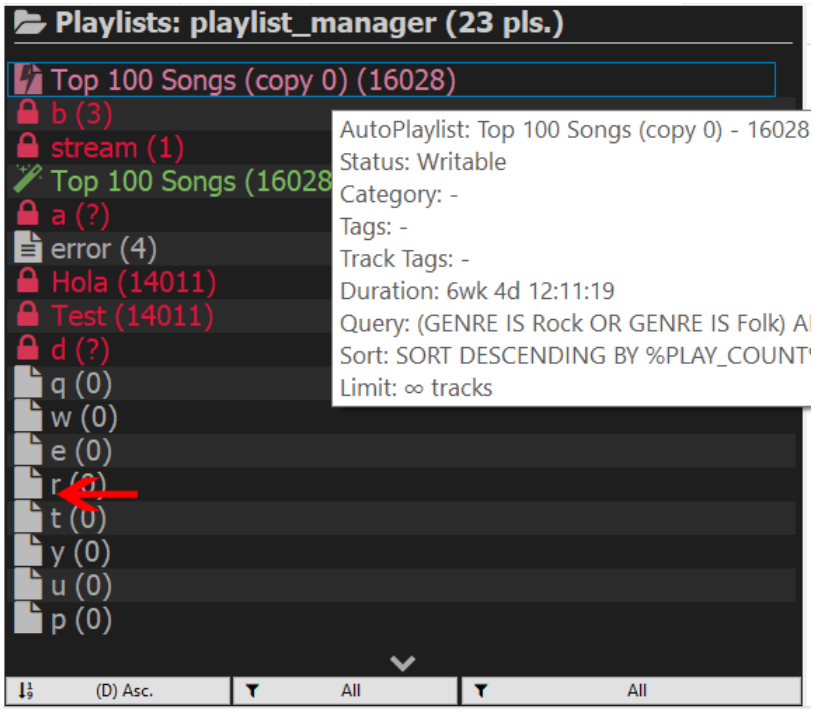


Figure 123: Playlist icons.

23.4 Playlist Icons

Playlist icons may be set to use any of the available icons on Font Awesome (v4.7.0 or greater). Predefined presets are already set for all the playlist formats [V].

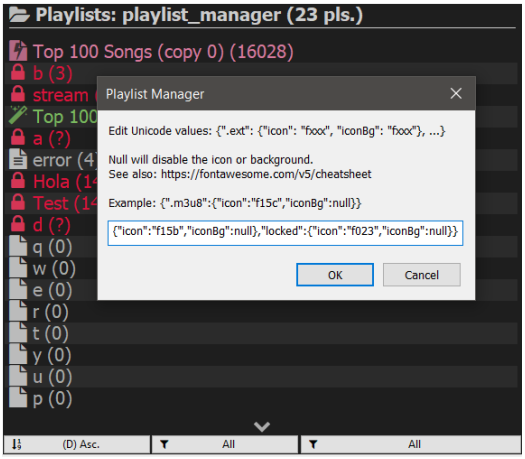


Figure 124: Presets for the default icons.

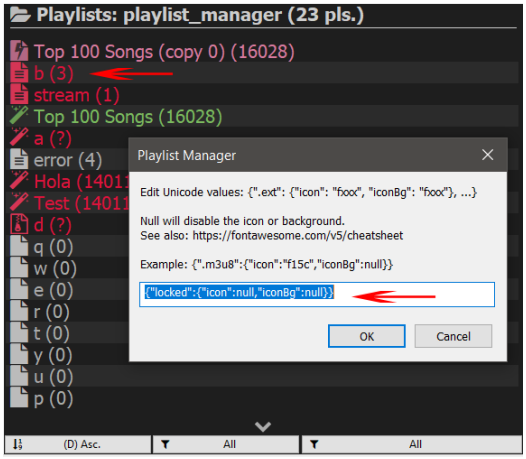


Figure 125: Locked icon disabled.

Icons not being set will use the default preset, therefore, to override the default icons, they must be set on the on the panel. To **disable a icon** for a single format it may be **set to null**. For ex. **nullifying the locked icon, playlist will use the playlist format icon as fallback**, nullifying both, will show none.

Note the formatting follows JSON guidelines, check WikiPedia for more info. Only keys (extensions) written will be used and override the default ones. The values are expected to be unicode values found on Font Awesome cheatsheet. The first time the configuration is opened will be filled with the default preset (which can be used as a guide).

Additionally, all icons may be hidden or shown with using the global switch if desired:

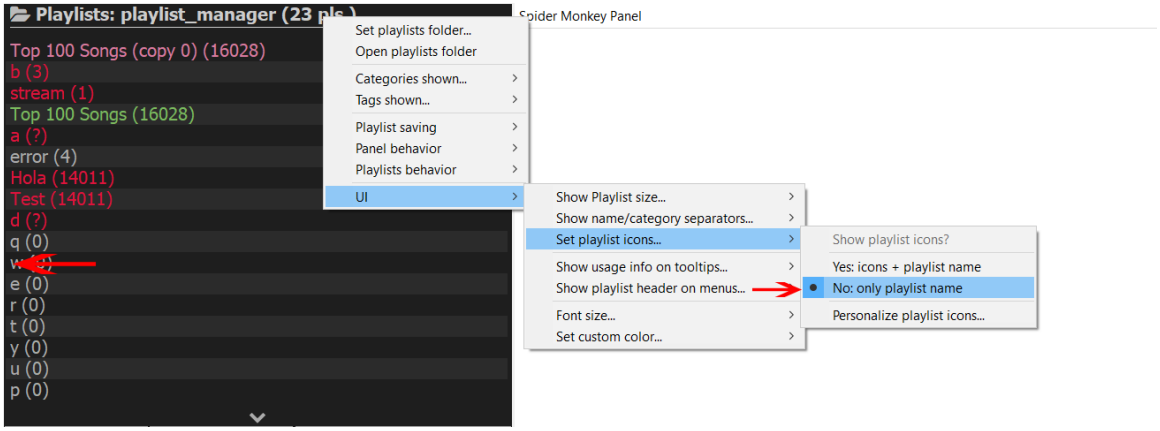


Figure 126: Hiding playlist icons.

23.5 Playlist action shortcuts

Different Playlist actions [18] may be associated to the 3 available keyboard modifiers when Left or Middle Clicking on any playlist of the list. i.e.:

- Ctrl + Left/Middle Click
- Shift + Left/Middle Click
- Ctrl + Shift + Left/Middle Click
- Double Left Click
- Single Midle Click

It should be noted that only Left Click actions are set by default on a newly installed panel, Middle Click actions are available although not set by default (except multiple selection). Available actions, which can be bound to any of those mouse events, are summarized below:

- None.
- Load / show playlist (defaults to Ctrl + L. Click).
- Send selection to playlist (defaults to Shift + L. Click).
- Clone playlist in UI (defaults to Ctrl + Shift + L. Click).
- Recycle playlist.
- Lock/unlock playlist file.
- Lock/unlock UI playlist.
- Multiple selection (defaults to M. Click).

Mouse actions -if available- are shown following the tooltip configuration [22.8]. Whenever a mouse event is set to none, no action is performed and entry will be hidden in tooltip.

24 Quick-searching

It's possible to jump directly to a playlist by pressing keys according to the sorting method. Full word searching is allowed, but the UI will continue jumping to the first match of the current string typed (i.e. if you want another match, keep typing the entire name). Quick-searching follows the currently sorting method set:

- By name: matches the playlist name. [alphanumeric]
- By category: matches the playlist category. [alphanumeric]
- By tags: matches the first value of all playlist's tags. [alphanumeric]
- By size: matches the playlist size (tracks). [numeric]
- By date: fallback to name. [alphanumeric]

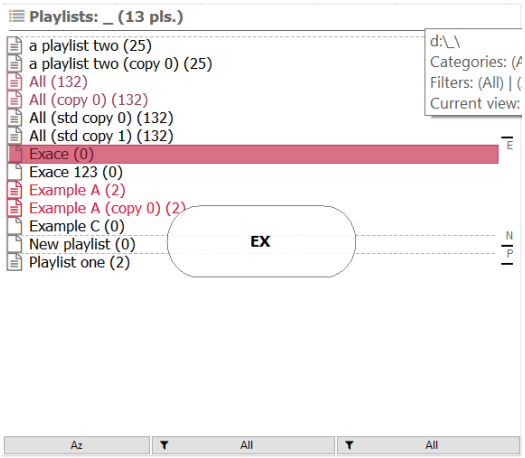


Figure 127: Successful search.

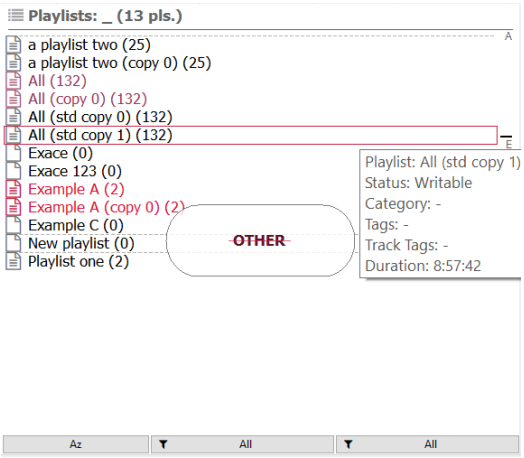


Figure 128: No matches.

To use it, **the panel must be on focus. If pressing a key with mouse over the panel does not initiate quick-searching, then click on the panel first and retry**. Note left/right clicking may open a menu on the panel, instead of that, **middle clicking may be used to grab focus anywhere on the panel without interacting with it**.

25 In panel popups

At startup, while caching the library item's paths [X] or at other points where the panel is processing asynchronously, the panel may get blocked to further usage while showing an animation.

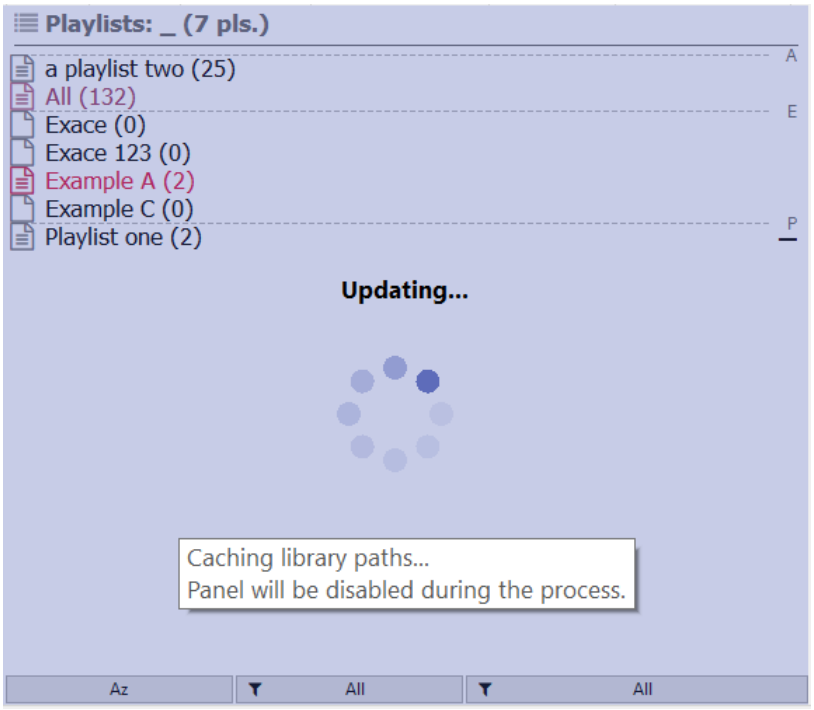


Figure 129: Animation popup while caching the library.

The mouse can be moved over the panel and a tooltip related to the current action being processed will be shown giving more info. Quick-searching [24] animation is also an specific example of this feature.

26 Scrolling

When the number of playlists exceeds the number of rows present on the panel, scrolling is allowed using multiple shortcuts:

- Mouse wheel (single rows).
- Up / Down keys (single rows).
- The scrolling arrows on the UI (single rows).
- Av Pag / Re Pag keys (entire pages).
- Home / End keys (go to top / bottom).

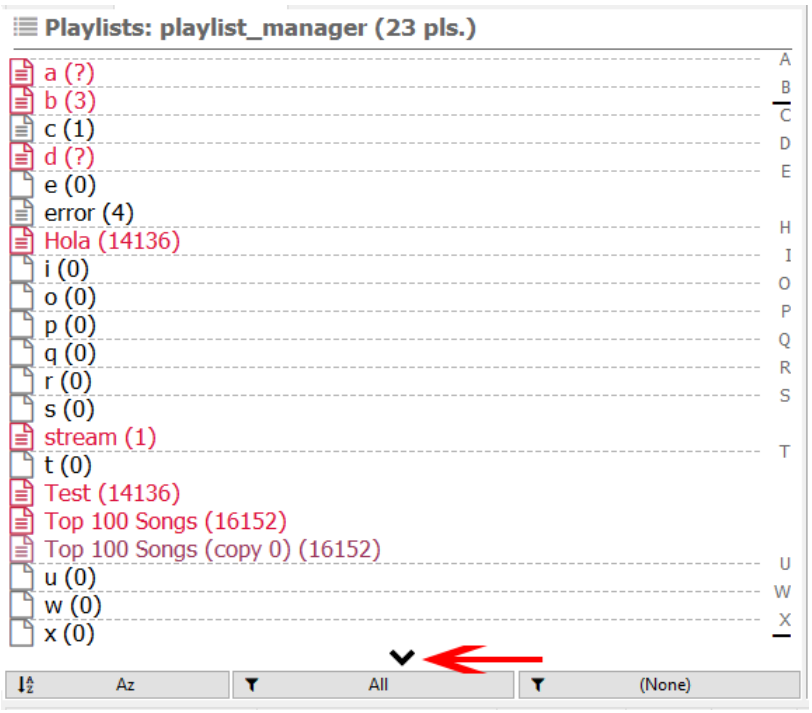


Figure 130: Scrolling arrow.

Part V

Playlist formats

- Writable formats: .m3u, .m3u8, .pls, .xspf and .jspf (XSPF json version).
- Readable only formats[52]: .strm, .fpl, Auto-Playlists and .xsp (Smart Playlist).

In general, writable formats work with more features than their readable only counterparts[53]. This is specially true for .fpl playlists, which are pretty limited on all aspects[54]. Therefore, the use of .m3u8 or .xspf playlists is preferred over other formats (.fpl or .pls) whenever it is possible.

For query-based formats, like Auto-playlists and Smart Playlists, there is obviously no 'limitation' despite being read only formats... since their aim involves creating playlists on the fly, not a list of files and paths.

27 Playlist metadata

27.1 Lock state

Playlist files may be locked or not. Locked files are considered read only for all purposes and therefore never rewritten unless forced to do so [by the user]. Native foobar playlists files (.fpl) are locked by default.

27.2 [Playlist] Tags

Playlists may have multiple tags, i.e. keywords for informative purposes. There are some special purpose tags which are associated to some actions performed by the manager automatically as soon as it loads a playlist with such keywords.

- Playlists may be tagged with 'bAutoLoad', 'bAutoLock' or a custom set of tags (for arbitrary purposes).
- 'bAutoLoad' makes the playlist to be loaded within foobar automatically (on the UI). Meant to be used on remote servers with online controllers.
- 'bAutoLock' locks the playlist as soon as it's loaded on the panel.

[52] For all purposes, locked playlists may be considered into this category, no matter their extension.
[53] There are some more exceptions to this rule. For example, .m3u8 playlists have more features than .pls ones.
[54] Neither exporting [14], nor additional tools [16] work with them. And obviously, since they are a readable only format, no new tracks may be added to them using the manager. Auto-Playlists are an exception to the latter, since tracks are 'calculated' at loading time.

27.3 Track tags

["tagName":"tagValue"]

Playlists may have multiple track tags, i.e. tag presets which are meant to be applied to the tracks within the playlist. Note this has nothing to do with the [Playlist] Tags which are just keywords. Allows multiple conditions (must follow json format) [VIII] where the tag value(s) can be any of the following:

- Foobar2000 Title Format expressions (or %tags%)[55].
- JavaScript functions (defined at 'helpers\helpers_xxx_utils.js'), prefixed by 'JS:'.
- Value (string or number).

27.4 Category

Playlists may have a single category for easy classification or which can be used as virtual folder. For informative purposes or arbitrary keywords use [playlist] tags instead]27.2].

27.5 UUID

Suffixes added to the playlist names to separate them from non tracked playlists when loaded in foobar. Some also allow some level of name duplication.

- Invisible Unicode chars plus (*)[56]
- (a-f)[57]
- (*)[58]
- Or none[59].

[55]https://wiki.hydrogenaud.io/index.php?title=Foobar2000:Titleformat_Reference

[56]Allows the highest degree of name duplication. For ex. multiple playlists with name 'Summer' would have different invisible UUID's... Experimental feature.

[57]Allows some degree of name duplication. The UUID will be visible along the name, thus being less useful than the Unicode version... but more stable.

[58]Duplication is not allowed but serves as a indicator of playlist being tracked by the manager.

[59]The only way to know if the playlist is tracked by the manager is by looking at the manager panel and checking the loaded or now playling indicators[21].

28 Playlist features table

Fully supported (X), partially supported (comment), not supported (empty) or not applicable (-):

Feature	M3U	PLS	STRM	XSPF	Foobar (FPL)	Foobar UI
Relative paths	X	X	X	X		
Subsongs (iso)	X			X	X	X
Playlist metadata	X			X	JSON	JSON
Tracks metadata	X	Title		X	X	X
XBMC compatibility	X	X	X	-	-	-
Plain text format	X	X	X	X		-
Exporting (+metadata)	X	X	X	X	File+JSON	

Table 2: Standard playlists.

Features	Auto-playlists	Smart Playlists (XSP)
Playlist Metadata	JSON	JSON
Foobar queries	X	Limited
XBMC queries		X
XBMC compatibility		X
Plain text format		X
Exporting (+metadata)	JSON	File+JSON

Table 3: Query based playlists.

- **Relative paths**: path to files may be saved as relative paths; for portable disks, network folders, etc.
- **Subsongs (iso):** tracks within an ISO file (DVDs, SACDs, ...) may be used, not pointing to the entire file but only the relevant track. This feature is only relevant within Foobar2000 context (since other players don't support it).
- **Playlist Metadata:** additional metadata associated to the playlist and used by the manager. This data may be sabed within the playlist file or on an external json database.
- **Tracks metadata:** tags like Title or Artist for every track are stored inside the playlist. Some formats may only store partial metadata.
- **XBMC compatibility:** playlist is compatible (readable, writable, ready to be exported,...) with other systems like Kodi, XBMC, etc.
- **Foobar queries:** playlist makes use of Foobar2000's queries capabilities, not pointing to specific tracks but created on demand. Partial support limits its use to an specific subset of queries or functions.
- **XBMC queries:** playlist makes use of XBMC's queries capabilities (equivalent to foobar queries functionality). Usage is not limited to Foobar2000's context in this case but also available on XBMC's systems. In many cases, XBMC queries have a Foobar query counterpart, but not all rules have an equivalence [49].
- **Plain text format:** playlist can be easily opened using any text editor.
- **Exporting (+metadata):** playlist can be exported to other programs preserving all their metadata. Some formats may require an additional JSON file for the metadata.

29 .m3u & .m3u8

M3U (MP3 URL) is a file format for a multimedia playlist. Although originally designed for audio files, such as Flac, it is commonly used to point media players to audio and video sources, including online sources, without distinction. There is no formal specification for the M3U format, it is a de facto standard.

An M3U file is a plain text file that specifies the locations of one or more media files. The file is saved with the 'm3u' filename extension if the text is encoded in the local system's default non-Unicode encoding (e.g., a Windows code-page), or with the 'm3u8' extension if the text is UTF-8 encoded. Within the manager, for all purposes, files generated by it are equivalent since '.m3u' files are also UTF-8 encoded[60].

Each entry carries one specification. The specification can be any one of the following:

- An absolute local path-name; e.g., C:\My Music\Listen.mp3
- A local path-name relative to the M3U file location; e.g., Listen.mp3
- A URL; e.g., http://www.example.com:8000/Listen.mp3

Position	Description	Entries
Header:	Required if using Extended M3U.	#EXTM3U,#EXTENC[,...]
Track(s)	Track entries, arbitrary number allowed.	[#EXTINF,]file path or url
	...	

Table 4: M3U structure.

29.1 Extended M3U

The M3U file can also include comments, prefaced by the '#' character. In extended M3U, '#' also introduces extended M3U directives which are terminated by a colon ':' if they support parameters.

Directive	Description	Example
#EXTM3U	File header, first line	#EXTM3U
#EXTENC:	Text encoding, second line	#EXTENC:UTF-8
#PLAYLIST:	Playlist display title	#PLAYLIST:My Playlist
#EXTINF:	Track information: seconds and title	#EXTINF:256,Chateau Pop - Maniac

Table 5: Standard M3U extensions.

[60] Although this 'breaks the standard', it's indicated with the appropriate extended M3U directive so it should be totally safe. Also note the 2015 proposal for HTTP Live Streaming follows the same convention.

Since arbitrary comments can be prefaced by '#' and custom directives are allowed, the manager includes its own set of directives to support additional playlist metadata [27]:

Directive	Description	Example
#UUID	Playlist UUID [27.5]	#UUID: (*)
#LOCKED:	Lock state	#LOCKED:false
#CATEGORY:	Playlist category	#CATEGORY:Summer
#TAGS:	Playlist tags (sep by ';')	#TAGS:Celtic;Chill
#TRACKTAGS:	Tags to apply tracks (json) [27.3]	#TRACKTAGS:["Mood":"Chill"]
#PLAYLISTSIZE:	Playlist size (# of tracks)	#PLAYLISTSIZE:2

Table 6: Additional M3U extensions for playlist metadata support.

The use of Extender M3U along the additional custom directives allows the manager to make use of all features without restrictions. For ex. playlists may have UUIDs independently of their playlist name, categories may be used to filter the list, etc.

An example of a full .m3u8 playlist with relative paths[61] can be found below:

```
#EXTM3U
#EXTENC:UTF-8
#PLAYLIST:My playlist
#UUID: (*)
#LOCKED:false
#CATEGORY:Summer
#TAGS:Celtic;Chill
#TRACKTAGS:["Mood":"Chill"]
#PLAYLISTSIZE:2
#EXTINF:256,Chateau Pop - Maniac
.\Music\Big Retro Hits 90s\007. Chateau Pop - Maniac.mp3
#EXTINF:259,Jaki Graham - Round And Around
.\Music\Big Retro Hits 90s\004. Jaki Graham - Round And Around.mp3
```

[61] Playlist file would be at current folder and tracks within 'music' subfolder.

30 .pls

PLS is a file format for a multimedia playlist. Used with audio and video sources, including online sources, without distinction.

PLS is a more expressive playlist format than the basic M3U playlist, as it can store information on the song title and length (supported in extended M3U only)[62].

The format is case-sensitive and essentially that of an INI file structured as follows:

Position	Description	Entries
Header:	Always required. 'As is'	[playlist]
Track(s)	File entries, arbitrary number allowed. X equals entry number ...	FileX[,TitleX,LengthX]
Footer	Always required.	NumberOfEntries,Version

Table 7: PLS structure.

Key-value pairs are separated by '=':

Entry	Description	Example
[playlist]	Always required	[playlist]
FileX	Location of media file/stream.	File1=http://stream2.streamq.net:8020/
TitleX	Track title (optional)	Title1=My radio station
LengthX	Seconds (-1 equals indefinite) (optional)	Length1=-1
NumberOfEntries	Playlist size (# of tracks). Required	NumberOfEntries=3
Version	only a value of 2 is valid. Required	Version=2

Table 8: PLS entries.

Since the PLS format follows a pretty strict format, additional metadata like categories or UUID's can not be used with them (playlist name is the same than the filename). Switch to another format to make use of those features.

An example of a full .pls playlist with relative paths[63] can be found below:

```
[playlist]
File1=.\foobar2000\Big Retro Hits 90s\004. Jaki Graham - Round And Around.mp3
Title1=Round And Around
Length1=259

File2=.\foobar2000\Big Retro Hits 90s\007. Chateau Pop - Maniac.mp3
Title2=Maniac
Length2=256

NumberOfEntries=2
Version=2
```

[62]This is only true for basic M3U playlists usually found on the net. Within the manager context, M3U playlists are always richer in metadata and features since they make use of extended M3U and the additional custom directives.

[63]Playlist file would be at current folder and tracks within 'music' subfolder.

31 .strm

STRM is a file format for a streaming multimedia playlist. Used with online audio and video sources without distinction.

STRM is a simple plain-text playlist format without any metadata support, only including one line: the direct URL-link to the stream. The format is not case-sensitive and essentially a plain-text file:

Position	Description	Entries
Stream:	Always required. Only one value allowed.	URL

Table 9: STRM structure.

Only one value is allowed at the body:

Entry	Description	Example
URL	Location of media stream.	https://rockfm-cope.flumotion.com/chunks.m3u8

Table 10: STRM entries.

Since the STRM format follows a really limited structure, no metadata or multiple streams can be saved on a file. Switch to another format to make use of those features. In particular, most players than can read STRM files can also read M3U files, so the latter are preferred even if the playlist will only contain one URL[64].

An example of a full .strm playlist can be found below:

https://rockfm-cope.flumotion.com/chunks.m3u8

The manager will try to ensure STRM files follow the specs by checking the actual number of paths/links on the file on loading, throwing a warning if an error is found. Note blank lines are not necessarily supported by some players, so these playlist may be forced to be one-line sized, although he manager will skip blank lines if it finds one.

[64]For ex. Kodi: Internet video and audio streams

32 .xspf

XSPF is an XML-based playlist format for digital media (audio or video), intended to shared and played on different devices.

Traditionally playlists have been composed of file paths/links that pointed to individual items. XSPF encourages a new kind of playlist sharing called content resolution: based on metadata. An XSPF-compliant content resolver will open XSPF playlists and search a catalog for every title with <creator>, <album> and <title> tags, then localize the playlist to reference the available matching tracks.

XSPF follows a XML structure, allowing full metadata support (via built-in variables and/or extensions). The format is case-sensitive:

Position	Description	Entries
XML Header:	Always required.	<?xml version="1.0" encoding="UTF-8"?>
Playlist:	Always required.	<playlist version="1" xmlns="http://xspf.org/ns/0/">
Metadata:	Optional.	<title>example</title>
	...	
Track list:	Always required.	<trackList>
Track:	Any number.	<track><location>file:///song1.mp3</location></track>
	...	
Track list end:	Always required.	</trackList>
Playlist end:	Always required.	</playlist>

Table 11: XSPF structure.

The format allows some built-in playlist info and arbitrary meta values [27]:

Metadata	Description	Example
title	Playlist name	<title>example</title>
uuid	Playlist UUID [27.5]	<meta rel="uuid"> (*)</meta>
locked	Lock state	<meta rel="locked">true</meta>
category	Playlist category	<meta rel="category">Chill</meta>
tags	Playlist tags (sep by ';')	<meta rel="tags">Celtic;Chill</meta>
trackTags	Tags to apply tracks (json) [27.3]	<meta rel="trackTags">["Mood":"Chill"]</meta>
playlistSize	Playlist size (# of tracks)	<meta rel="playlistSize">817</meta>

Table 12: XSPF meta for playlist metadata support.

Other considerations to ensure XSPF conformance by the specs (also XSPF v1 Notes and Errata):

- Relative paths MUST be resolved according to the XML Base specification or IETF RFC 2396.
- MUST handle valid URIs.
- MUST handle failure of any one resource. If an xspf:track element cannot be rendered, an user-agent MUST skip to the next xspf:track element and MUST NOT interrupt the sequence.
- xspf:playlist elements MUST contain one and only one trackList element (may be empty).
- xspf:track elements MAY contain zero or more location elements, but a user-agent MUST NOT render more than one of the named resources.
- trackNum MUST be a valid XML Schema nonNegativeInteger.
- duration MUST be a valid XML Schema nonNegativeInteger.

Multiple tracks are allowed and each one may have any of these values (case sensitive):

Entry	Description	Example
location	URI encoded link/path to file.	<location>file:///song1.mp3</location>
title	Item name.	<title>Find the river</title>
creator	Artist.	<creator>R.E.M.</creator>
album	Album name.	<album>Automatic for the people</album>
duration	Item length, in milliseconds .	<duration>233000</duration>
trackNum	Item number.	<trackNum>12</trackNum>
identifier	Hash, MusicBrainz Id, ...	<identifier>161c7-ce73-4e97-9f0c-9f0e53</identifier>

Table 13: XSPF entries. For additional variables check XSPF Version 1.

As can be noted, the XSPF format allow the same degree of arbitrary playlist metadata than the M3U format while also setting an standard to the tracks' metadata: tags like album, track number or artists are enclosed in their own standardized variables and correctly identified instead of relying on an arbitrary format for the track's info that may change from player to player.

Content resolution is allowed via fuzzy matching using these tags to find matches on the library via queries (something Foobar2000 excels at). Therefore, whenever a track is not found on the library, the manager will try to find a match on the library using the track's metadata. Note this is a clear advantage over M3U playlists, since it allows full portability no matter the media paths[65]. The specification also enforces that non found items (after queries) should simply be skipped, thus partial loading of playlists is also allowed. In many aspects this is pretty similar to Auto-Playlists, although limited to the list of tracks and allowing only 1 result per item.

An example of a full .xspf playlist can be found below:

```
<?xml version="1.0" encoding="UTF-8"?>
<playlist version="1" xmlns="http://xspf.org/ns/0/">
    <title>example</title>
    <identifier>example</identifier>
    <attribution>XXX</attribution>
    <meta rel="uuid"></meta>
    <meta rel="locked">true</meta>
    <meta rel="category"></meta>
    <meta rel="tags"></meta>
    <meta rel="trackTags"></meta>
    <meta rel="playlistSize">817</meta>
    <trackList>
        <track>
            <location>file:///H:/music/12 - Find the river.mp3</location>
            <title>Find the river</title>
            <creator>R.E.M.</creator>
            <album>Automatic for the people</album>
            <duration>233000</duration>
            <trackNum>12</trackNum>
        </track>
    </trackList>
</playlist>
```

[65] At another devices the files may be in a different disk, folder structure or even named totally different... but as long as the metadata is the same, the playlist will be loaded fine.

33 .fpl

FPL is a binary file format for a multimedia playlist by Foobar2000. Used with audio and video sources, including online sources, without distinction.

It's a closed source format whose structure has not been shared publicly, although it's known it uses a binary format to store the metadata of the tracks included to greatly speed up its loading time within the program[66].

Although it looks as an improvement over plain text playlist formats, the 'closed source & binary' format requirements no longer holds true to offer short loading times. Plain text playlist formats may be used perfectly fine, as long as the files are matched with those on the library, without speed penalties[67]. This is the behavior followed by manager and it has been already reported to Foobar2000 developers [68] to properly implement playlist loading if desired. Loading speed penalties only happen when some items are not on library (whether they are external items or dead items) [16].

To allow additional metadata for .fpl playlists, considering the files are non-editable, an external json file is used [VIII]. The same applies to Auto-Playlists. The following keys-values pairs are used:

Entry	Description	Example
id	UUID	"id": " (*)"
name	Playlist name	"name": "example"
nameId	Playlist display name	"nameId": "example (*)"
extension	Playlist file extension	"extension": ".fpl"
path	Playlist file path	"path": ".\profile\playlist_manager\example.fpl"
size	Playlist size (# of tracks)	"size": 2
fileSize	Playlist file path	"fileSize": 20739
isLocked	Lock status	"isLocked": true
isAutoPlaylist	Is an Auto-Playlist?	"isAutoPlaylist": false
query	Auto-Playlist query	"query": ""
sort	Auto-Playlist sort(optional)	"sort": ""
bSortForced	Auto-Playlist sort forced?	"bSortForced": false
category	Playlist category	"category": "Summer"
tags	Playlist tags (sep by ';')	"tags": ["bAutoLoad","bAutoLock"]
trackTags	Tags to apply tracks (json)	"trackTags": ["Rating": 5]
limit	Playlist output size limit	"limit": 0

Table 14: FPL (and Auto-Playlist/Smart Playlist) json format.

Note all Auto-playlist related metadata is empty, the path points to the physical .fpl file and its file-size is cached[69]. Apart from those differences, it can be easily checked that all playlist metadata is present (the same it was in M3U format). In fact, all playlists are converted -for internal use- to this format within the code.

[66] Instead of loading the tracks and retrieving their metadata from the physical files.

[67] Which is one of the most common use-case of playlists within Foobar2000.

[68] Check Why m3u8 loading is so slow on hydrogenaud.io-

[69] For Auto-playlists, it would be the contrary. No physical file is associated and the query and sorting is used instead... but -essentially- they use the same format.

An example of a full .fpl playlist associated json file can be found below. In real files, every playlist would be concatenated to the same file, thus having multiple {playlists objects}, separated by comma (','), between the brackets [{...}, {...}]:


```
[
    {
        "id": " (*)",
        "name": "Example",
        "nameId": "Example (*)",
        "extension": ".fpl",
        "path": ".\profile\playlist_manager\Intercalate mix.fpl",
        "size": 51,
        "fileSize": 20739,
        "isLocked": true,
        "isAutoPlaylist": false,
        "query": "",
        "sort": "",
        "bSortForced": false,
        "category": "Summer",
        "tags": ["bAutoLoad","bAutoLock"],
        "trackTags": ["Mood": "Chill"]
        "limit": 0
    }
]
```


Note on such .json file (one per Playlist Manager panel), these playlists may be present along Auto-Playlists and Smart Playlists, since all those formats have a json counterpart for metadata purposes. Their identification relies on the extension, file path, 'isAutoPlaylist' boolean and/or query.

34 Auto-Playlists

Auto-playlists are virtual playlists -no physical file- intended to retrieve multimedia items by queries within Foobar2000. Other programs have similar features: Kodi and XBMC (Smart Playlists), Musicbee (Auto-Playlists), etc.

Structure and format is exactly the same than .fpl use-case, so check it for reference [33]. Specifics for Auto-Playlists are listed below:

Entry	Description	Example
extension	Playlist file extension	"extension": ""
path	Playlist file path	"path": ""
fileSize	Playlist file path	"fileSize": 0
isAutoPlaylist	Is an Auto-Playlist?	"isAutoPlaylist": true
query	Auto-Playlist query	"query": "ALL"
sort	Auto-Playlist sort (optional)	"sort": "SORT DESCENDING BY TITLE"
bSortForced	Auto-Playlist sort forced?	"bSortForced": true
limit	Playlist output size limit	"limit": 0

Table 15: Auto-Playlist json format changes.

Extension and path are empty, since there is no associated physical file. File-size is therefore equal to zero. 'isAutoPlaylist' boolean is true and the query related value must be filled. Sorting is optional.

An example of a full Auto-Playlist associated json file can be found below[70]. Note, in real files, Auto-Playlists may be mixed with other playlists:


```
[
    {
        "id": "",
        "name": "Entire Library",
        "nameId": "Entire Library",
        "extension": "",
        "path": "",
        "size": 132,
        "fileSize": 0,
        "isLocked": false,
        "isAutoPlaylist": true,
        "query": "ALL",
        "sort": "SORT DESCENDING BY TITLE",
        "bSortForced": true,
        "limit": 0,
        "category": "Summer",
        "tags": [],
        "trackTags": [],
        "limit": 0
    }
]
```


[70] For ex. if the panel is set to track 'H:\My Music\Playlists', then the playlist json file (at foobar profile folder) will be at '.\js_data\playlistManager_Playlists.json'.

35 .xsp -Smart Playlists-

.xsp format or 'Smart Playlists' are an XML-based playlist format on Kodi-like and XBMC systems, pretty similar to AutoPlaylists, which also retrieve multimedia items by queries.

XBMC's queries use their own format and tags, so they are not directly compatible with Foobar2000's queries. To be able to use these playlist, a full translation layer has been added between both query formats. This imposes some restrictions since most TF functions don't have an equivalence on XBMC's queries, they are mostly limited to standard queries without functions. For more info check [49].

XSP follows an strict XML structure without metadata support. The format is case-sensitive:

Position	Description	Example
XML Header:	Required.	<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
Playlist Type:	Required.	<smartplaylist type="songs">
Name:	Playlist name.	<title>Top Rock songs</title>
Match:	'all'/'one'	<match>all</match>
Rule:	Any number.	<rule field="genre" operator="is"><value>Rock</value></rule>
		...
Limit:	Optional.	<limit>100</limit>
Sorting:	Optional.	<order direction="descending">lastplayed</order>

Table 16: XSP structure.

Multiple rules are allowed (matching all -'all'- or any of them -'one'-), and each one may have multiple values (case sensitive):

Entry	Description	Example
Rule	Specifies tag field and operator.	<rule field="genre" operator="is">
Value	Tag value to match.	<value>Rock</value>
	...	
Rule closing	Closes value list.	</rule>

Table 17: XSP rule entries.

Within the same rule, multiple values are evaluated following an 'OR' pattern. i.e. 'value = Rock, Value = Folk', it translates into something like 'Value = Rock or Value = Folk'. On the contrary, rules are evaluated following the global match attribute ('all'/'one').

Within a rule, the list of operators allowed to apply on its values is:

Entry	Description
contains	True if the field contains the value as a substring.
doesnotcontain	True if the field does not contain the value as a substring.
is	True if the field matches the value exactly.
isnot	True if the field does not match the value exactly.
startswith	True if the start of the field matches the value.
endswith	True if the end of the field matches the value.
lessthan	True if the content of the field is less than the value.
greaterthan	True if the content of the field is greater than the value.
after	True if the content of the field comes after the value. Date fields.
before	True if the content of the field comes before the value. Date fields.
inthelast	True if the field falls in the last range specified by the value. Date fields.
notinthelast	True if the field does not fall in the last range specified by the value. Date fields.

Table 18: XSP operators.

And the list of tag fields which work within XBMC systems are:

Fields	
genre	lastplayed
source	rating
album	userrating
artist	comments
albumartist	moods
title	playcount
year	path
time	filename
tracknumber	

Table 19: XSP tag fields.

An example of a full .xsp playlist can be found below:

```
<?xml version="1.0" encoding="UTF-8" standalone="yes" ?>
<smartplaylist type="songs">
    <name>Top 100 Songs</name>
    <match>all</match>
    <rule field="genre" operator="is">
        <value>Rock</value>
        <value>Folk</value>
    </rule>
    <rule field="year" operator="greaterthan">
        <value>1969</value>
    </rule>
    <order direction="descending">playcount</order>
    <limit>100</limit>
</smartplaylist>
```

The Foobar2000 query equivalence would be: '(GENRE IS Rock OR GENRE IS Folk) AND (DATE GREATERTHAN 1969)'.

For the associated .json file, to store metadata, structure and format is exactly the same than .fpl use-case, so check it for reference [33]. Specifics for Smart Playlists are listed below:

Entry	Description	Example
extension	Playlist file extension	"extension": ".xsp"
path	Playlist file path	"path": ".\profile\playlist_manager\example.xsp"
fileSize	Playlist file path	"fileSize": 20739
isAutoPlaylist	Is an Auto-Playlist?	"isAutoPlaylist": false
query	Auto-Playlist query	"query": "ALL"
sort	Auto-Playlist sort (optional)	"sort": "SORT DESCENDING BY TITLE"
bSortForced	Auto-Playlist sort forced?	"bSortForced": false
limit	Playlist output size limit	"limit": 100

Table 20: Smart Playlist json format changes.

Extension and path are present, contrary to Auto-Playlists, since there is an associated physical file. File-size is therefore non zero. 'isAutoPlaylist' boolean is false and the query related value is filled with the foobar2000 equivalent query. Sorting is optional and 'bSortForced' is never used. Limit is only used by Smart Playlists and may be set to configure how many tracks will be sent to the playlist after using the queries.

An example of a full Smart Playlist associated json file can be found below[71]. In real files, Smart Playlists are mixed with .fpl playlists and Auto-Playlists... the only distinction being the 'isAutoPlaylist' boolean value and extension:


```
[
    {
        "id": "",
        "name": "Top Rock songs",
        "nameId": "Top Rock songs",
        "extension": ".xsp",
        "path": ".\profile\playlist_manager\example.xsp",
        "size": 100,
        "fileSize": 20739,
        "isLocked": true,
        "isAutoPlaylist": false,
        "query": "(GENRE IS Rock) AND (%RATING% GREATER 3)",
        "sort": "SORT DESCENDING BY %RATING%",
        "bSortForced": false,
        "limit": 100,
        "category": "Top",
        "tags": [],
        "trackTags": []
    }
]
```


Note the **query and sort variables store the already translated Title Format expressions** instead of the original XBMC ones.

[71] For ex. if the panel is set to track 'H:\My Music\Playlists', then the playlist json file (at foobar profile folder) will be at '.\js_data\playlistManager_Playlists.json'.

36 .ui -UI-only Playlists-

UI-only playlists are simply virtual placeholders for those playlists within the UI not being tracked by the current Playlist Manager panel[72]. They are not files and are not saved in any way to either separated playlist files or json, they are just an internal format.

Their only aim is showing an entry in the list view pointing to the playlist within Foobar2000 UI. This allows some non-advanced actions like fast switching to it, renaming, etc. [18]. In other words, using the manager as an standard playlist organizer like other components[73] (without the advanced functionality of having a physical file associated).

UI-only playlists may be used/shown along the other formats, have their own custom color [21] and are refreshed whenever a playlist is loaded/created/deleted. Their size is also shown along the name, but it's only updated according to the auto-saving interval configuration.

This is a comprehensive list of the features allowed for UI-only playlists:

- Show binded playlist (and its shortcut counterpart[18]).
- Send selection to playlist (and its shortcut counterpart[18]) [16.1.2].
- Rename...
- Delete [8.8].
- Clone as... [16.1.3].
- Export and covert tracks to... [14.3].
- Aditional tools [14.3]: External items, dead items and duplicated items.

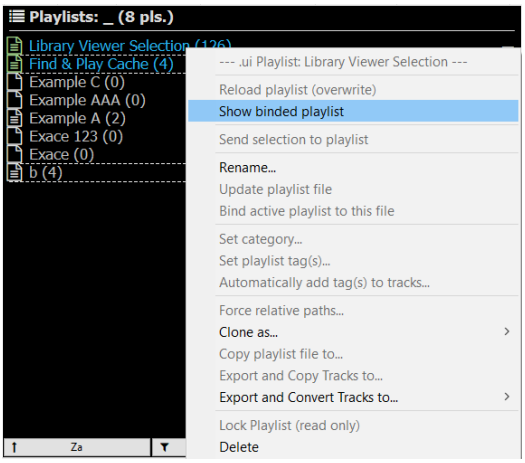


Figure 131: UI-only playlist menu.

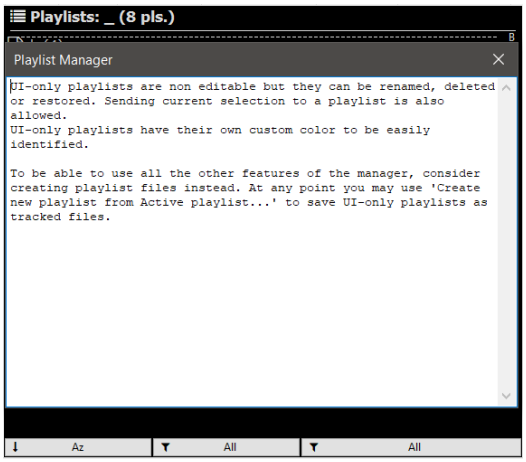


Figure 132: Warning popup.

[72] They may be associated to a file on another instance though.
[73] For ex. Playlist Organizer (foo plorg)

To associate the active UI-only playlist to a file, it may be either bound to a selected file [8.7] or to a new playlist file [8.5]. Then the virtual UI-only entry will get replaced with the physical one. Note this is different to cloning, since that option just creates a copy with a new name. Trying to rename an existing file to match an UI-only playlist's name will fail warning about no duplication allowed.

UI-only playlists are disabled by default on the manager but may be enabled on the config menu.

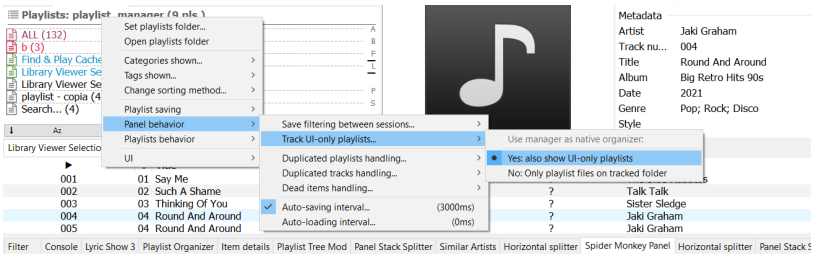


Figure 133: Enabling UI-only playlists on list view.

When UI-only playlists are enabled on the manager, the list view filters [22] are expanded to also consider these playlists in the 'Playlist type' and 'Extension' filters.

Part VI

Other scripts integration

This is a non comprehensive list of other Spider Monkey Scripts or plugins which may be used along the manager or whose features are designed to work together:

- Playlist-Tools-SMP:
  * **Random Pools:** Pools may use tracks from playlists files tracked by the manager, not requiring to have the playlists loaded within foobar. i.e. Random Pools component-like playlist creation, using not only queries as sources, but also other playlists or playlists files. See [48].
  * **Playlist Revive:** Finds and replaces dead items on loaded playlists or selection. Meant to be used along dead items checks on playlist files [16.2.2]. First check all playlist files, then load those with dead items and use Playlist Revive.
  * **Duplicates and tag filtering:** The manager allows to report playlists with duplicated items, but it's limited to entries with same path. This tool expands Foobar2000 native functionality of removing duplicates, allowing to find duplicates by tags (for ex. any track with same Title - Artist).[74]
  * **Import track list:** Takes a plain text list of tracks (for ex. Title - Artist) and finds matches on library to create a playlist. Meant to be used for playlist importing when the track list does not follow an standard format or there are no paths provided[75]. Instead of sharing a list of files, list of tracks may be used which work universally no matter your configuration. Non found items are simply discarded.

[74] A limited functionality version has been included which applies when cloning Auto-playlists if configured to do so [13.3].

[75] Technically that is not a playlist. But note playlists with relative paths may easily be considered a track list as long as you discard the '.\' part. In other words, a plain-text list can be retrieved from playlists in many cases.

Part VII

Online controller (ajquery-xxx)

37 Features

Playlist Manager can be integrated with foo_httpcontrol to be used within a web browser. A compatible template is required, like ajquery-xxx, specifically designed to integrate with Spider Monkey Scripts. Multiple panels may be integrated at the same time on the controller.This is a non comprehensive list of features available on the online controller:

- **View playlists:** list of available playlists in all panels, along basic metadata.
- **Load playlists** [8.6]
- **Clone playlists in UI** [16.1.3]
- **Lock / unlock playlists** [8.9]
- **Send selection to playlist** [16.1.2]
- **Delete / restore playlist** [8.8]
- **New empty playlist / from active playlist** [8.5]
- **Manual refresh** [17]

38 Installation

First follow the instructions of foo_httpcontrol and ajquery-xxx; the first must be simply installed as any plugin and the latter must be extracted within foo_httpcontrol templates folder.

Once both pieces are installed, enable "Dynamic menus" on the integration section of the header menu [18] along the other requisites (**foo_run_main:**). See [19] for more info.

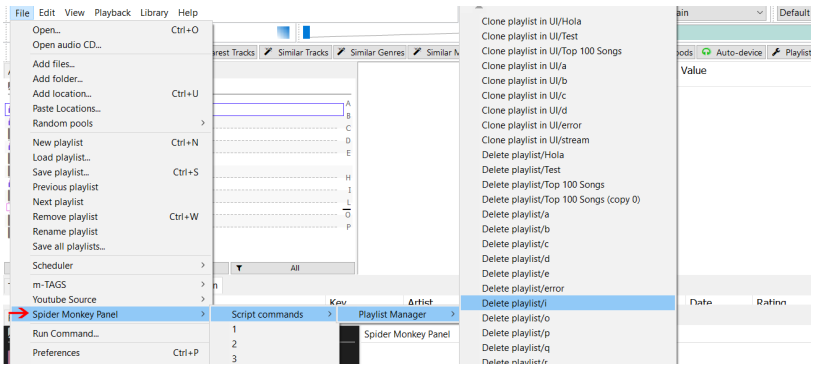


Figure 134: Menu at '*File\SpiderMonkeyPanel\ScriptCommands*'

Part VIII

FAQ

- **Writing playlists to files fails due to permissions problems.** Use something like this in a CMD instance (replace path and disk):

  attrib -r "DISK_LETTER:\YOUR_PLAYLIST_PATH*.*" /D /S

  ⇓⇓⇓

  attrib -r "D:\My playlists*.*" /D /S

- **Native foobar playlists (.fpl) can't be edited. Why?** Due to their closed source nature [33]. Unlock them first on the manager and format will be changed to allow editing.

- **How to use native foobar playlists (.fpl) without changing their format?** .fpl playlists are locked by default, so they will never be auto-saved (and thus reformatted) without user intervention. Just save all your foobar playlists on the tracked folder and load them when needed using the manager. Whenever you make a change on any of them, re-save it manually using main menu (File/Save playlist...). This way the native format is maintained while some neat features are still available for use (not cluttering the UI with all playlists on tabs, categories, tags, etc.).

- **How to use streaming playlists (.strm) without changing their format?** .strm playlists are locked by default, so they will never be auto-saved (and thus reformatted) without user intervention. If you wish to rewrite its format, unlock the playlist or force an update [8].

- **To create/track a playlists folder in the same path Foobar2000 resides in, relative paths may be used (.\playlist_manager\server\):** Note this will allow the manager to work properly on portable/network installations where the drive letter or absolute path changes.

- **To create/track a playlists folder in the same path Foobar2000 resides in, relative paths may be used (.\playlist_manager\server\):** Note this will allow the manager to work properly on portable/network installations where the drive letter or absolute path changes.

'.\playlist_manager\server\'

```
[foobar2000]
└─ [playlist_manager]
   ├─ [server]
   │  ├─ My playlist.m3u8
   │  └─ ...
   └─ [other]
      └─ summer.m3u8
```

.\profile\playlist_manager\server\'

```
[foobar2000]
└─ [profile]
   └─ [playlist_manager]
      ├─ [server]
      │  ├─ My playlist.m3u8
      │  └─ ...
      └─ [other]
```

- **Sending tracks to playlists easily:** 'Send selection' playlist menu entry and/or shortcut [18] allows to send currently selected tracks to the desired playlist without opening or switching to it [16.1.2]. It may be used to easily manage the playlists without cluttering the UI. Also **duplicates are checked, so adding multiple times the same tracks is not allowed; duplicates are simply skipped**.

- **Native playlists are too limited in features:** The use of M3U playlists is preferred since it allows the full use of all features. This is by design, and nothing can be done unless the format becomes open source or Spider Monkey Panel supports directly editing/saving them. Nothing is lost using M3U playlists, since they load as fast as native playlists when using the manager. .pls format is more limited in the metadata side.

- **Playlist metadata is lost on format switch**: Since only .m3u8 supports the full set of metadata and features, converting those playlists to .pls necessarily implies discarding of not supported metadata. Converting those playlists back to .m3u8 format will not restore that info once is lost!

- **What's json?** It's a standard file structure. Check https://en.wikipedia.org/wiki/JSON for more info.

- **What's asynchronous execution?** Execution of some code done on the background (usually on iterative steps), thus not blocking the UI on the process. For ex. external playlist files loading on native Foobar2000 [8.6].

- **Can't edit or update a playlist:** Check the playlist status. It's probably locked, unlock it to be able to make changes.

- **How to "disable" the selection rectangle?:** Just change its color to the one set at the background. It has been coded to be omitted in that case.

- **How to duplicate a playlist?:** Load the playlist you want to duplicate, then use the list menu [18] to "Create new playlist file from active playlist" [8.5]. Just input a different name and as result it will be duplicated using the new name. Alternatively use cloning [16.1.3].

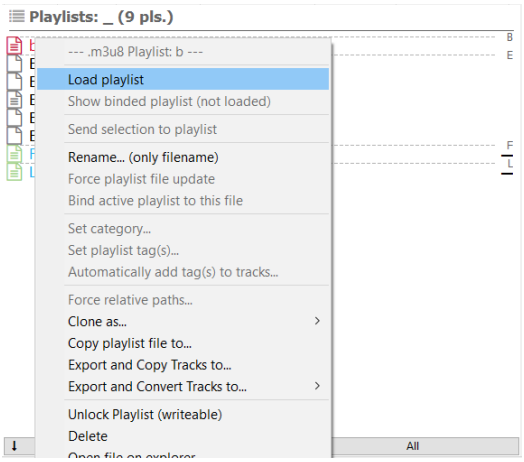


Figure 135: Load desired playlist.

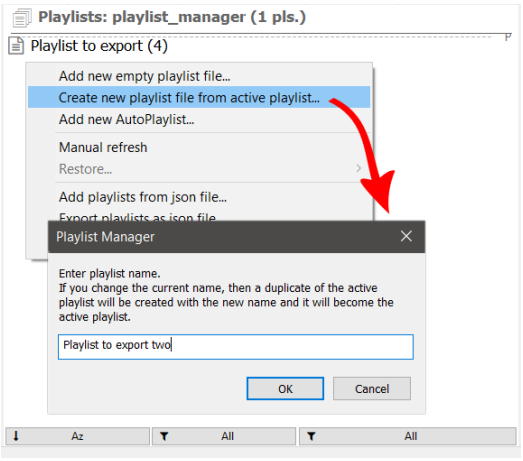


Figure 136: And create a new one from it.

- **Some items in a playlist are not being found on library although they are present:** That probably points to code-page detection errors by SMP on reading. UTF8 files without BOM may be incorrectly identified and thus not parsed as intended. Consider adding BOM [8.5] or using M3U formats which have a flag to notify their code-page [29]. .m3u8 files are always read as UTF8 files by design (whether they have BOM or not). These tips apply to playlists created within the playlist manager but for files generated by other software similar considerations may apply. Unix users may prefer non BOM files and simply use to M3U formats in any case. Multiple code checks have been added to minimize these situations in any case.

- **.pls playlist files have problems with unicode chars:** Since .pls does not have a field which specifies the encoding, it must be guessed using SMP heuristics (which is known to fail). They should work fine using UTF8 with BOM, but otherwise it depends on luck. Consider using BOM or switching to M3U formats instead.

- **Some playlist load right but track names have tons of rubbish chars:** Read previous comments about UTF8 encoding.

- **How is Auto-playlists size calculation processed at start-up?:** Contrary to native Foobar2000, Auto-playlists are calculated fully asynchronously since the playlists are not loaded in the UI. Therefore there is no need to finish the process to show the main program or panel and it can be done in the background. In other words, enabling Auto-playlist size updating at startup [23.2] is essentially performance-wise free at start-up in most cases.

- **How is Auto-playlists automatic tagging processed at start-up?:** Idem [12].

- **Are playlist exporting options asynchronous? [14]:** Due to its nature, copying just the playlist files is instantaneous. The copy and convert [with tracks] options takes some time since they involve bigger media files. Conversion is done using the Fobar2000 native processing, so it's done asynchronously on the background with a popup showing the progress. Copying tracks is done via SMP scripting and there are 2 ways to execute it, synchronous or asynchronous, which can be configured at the header menu [18]. The first has the advantage of checking that the files have been copied [76] (so it only ends when the last file has been processed), while the latter performs the copy on the background -whitout blocking the UI- and does not warn when the process ends.

- **Are files overwritten when using the exporting tools? [14]** Short answer: No. Long answer: Copying only the playlist file explicitly disallows overwriting an existing file, so if you try it fails. Copying the playlist file along its tracks allows to overwrite the playlist file (because it's in fact an edited version with relative paths which may need updating), but tracks are not overwritten. They are skipped instead... so if you export playlists -to the same path- with the same tracks contained in some of them, they are only copied once. The conversion option depends on the way you set the converter preset, since it may be set to ask/overwrite/skip. **Note you can make use of the skipping feature to easily export tons of playlists while also optimizing disk space, since physical files may be shared between playlists.** Obviously this only works if the filename mask is set properly, if you use something like %list_index% then every playlist will have its own files.

- **How to automatically tag tracks on a playlist? What input is expected at the popup?** Some working examples with descriptions can be found at [12].

- **The playlists created by the manager don't have the format shown in the examples [29, 30]:** Check the installation process and paths. Take a look at the file named '*_TIPS and INSTALLATION.txt*' which should be found along this readme.

- **The manager doesn't work as expected at random instances:** idem.

- **The manager shows a loading animation and doesn't allow further interaction:** this is expected behavior while the cache is being (re)built [X].

[76] Not talking about verifying checksum against the original file, but ensuring there have not been obvious errors.

- **Export and convert options may be used to export playlists with modified paths:** closing the tracks conversion window, the only file output will be the playlist. This trick may be used to recreate playlists with different drives, update them with different folder structure (Artist\Album\..., Artist\Date - Album\...), etc. without having to re-convert the media files. Have in mind this is usually much faster, you can use file operations to adjust media paths (moving files to follow different folder structure is faster than recreating them again).
- **The manager warns about some playlists having duplicated names:** either rename the reported playlists using the manager (they must be unlocked first) or using a text editor.
- **Why playlist names duplication warning is configurable?** Because disabling it allows to save the same playlist to 2 different files which may have different formats, i.e. saving a playlist as .m3u8 and .xspf at the same time.
- **The manager warns about size mismatch on header:** the playlist needs to be rewritten to fix it, clone it with the same format -example (copy 0)-, delete the old one and rename the new playlist back to the original name -example-.
- **UI-only playlists may be easily converted to a playlist file** either using 'Create new playlist file from active playlist' [8.5] or binding them to an already existing file [8.7].
- **Format friendly actions** are those which don't involve changing the original format [V] of the playlist to the default one set at the panel [8.4]. For ex: playlist loading into the UI [8.6], renaming, deleting and restoring [8.8], locking [8.9], sending selection to playlist [16.1.2] and any playlist-metadata edits [27]. Obviously, cloning or exporting tools don't change the original file, but may involve changing format of the output file.
- **Beware of loading and editing playlists within the UI if their original format must be preserved:** sending selection may be used instead [16.1.2] which ensures the original format is used. On the contrary, auto-saving or manually saving a playlist file will always use the default format set on the panel [8.4].
- **How to load a playlist as read-only file (without locking it)?:** use cloning [16.1.3] to create a duplicate as an UI-only playlist.
- **Text color changes automatically after changing the background color:** this is a feature of the UI, which tries to use the opposite color to the background to improve readability by default. It may still be changed using a custom color [23].
- **I haven an error after installation:** check known problems [XI] and bug procedure [XII].
- **I have found a possible bug not related to a wrong installation:** please, create a new issue at github: https://github.com/regorxxx/Playlist-Manager-SMP after checking this [XII].

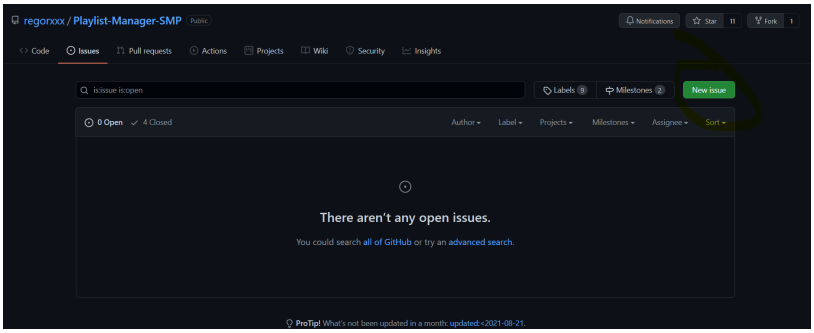


Figure 137: Opening a new issue at github.

Part IX

Advanced tips

39 Sharing manager files

- As noted on [8], **Auto-playlist's json file is saved using the tracked playlist folder name**. Instead of using an arbitrary UUID to avoid collisions between multiple panels, this can be used to **easily share playlists between different foobar instances**. Just create SymLinks[77] or use some cloud syncing tool (like Dropbox) to easily share the same playlists when tracking the same folder on different foobar instances or panels. Note only the folder name is used, so it would work even on shared network folders.

- **To share all playlist, sync all files in the tracked folders (which must share the same name) plus the json file.**

- The same principle can be used to **only share Auto-playlists**. If 2 manager instances track folders with the same name, even if the physical files are different, the json file can still be Sym-Linked or synchronized. Every instance would have different [standard] playlists but the same Auto-playlists.

- **Note .fpl playlists' metadata is also saved in the json file**, so beware of only sharing the json between multiple manager instances for Auto-playlist syncing (and not physical playlist). **.fpl metadata may be lost** if an instance doesn't have associated .fpl files, thus being lost globally.

40 Multiple views

- Following the same principle, it's possible to have **multiple panels tracking the same folder in the same foobar instance**. It may be used to have different filters enabled at the same time. For ex. one view for Auto-playlists only and another for physical playlists.

- Since **configuration is panel dependent, adjust as needed to optimize startup or responsiveness according to the contents**. For ex. if used to show Auto-playlist and physical playlists in 2 different panels, it would be advisable to completely disable any Auto-playlist processing on the latter (size updating, auto-tagging, etc.) and auto-loading on the first (since physical files don't need to be up to date).

- **Categories may be used as virtual folders, even if all playlists are in the same physical folder**. Note every playlist can only have one category at the same time, so cycling the categories allows to easily see different 'virtual sub-folders'. Double clicking on the header allows to easily do that. Alternatively, multiple panels tracking the same folder can be used to show 2 views of the same physical folder but with different categories filters, again working as sub-folders.

[77] Symbolic links are virtual links created by the OS which may be used to have multiple virtual files pointing to the same physical file. It's similar to a shortcut, although the extension doesn't change in this case... and the file properties are those of the original one.

41 Portable 'plug&play' installation

- **Real portable installations (i.e. on a external drive, network installations, etc.) may need to track playlist folders using their relative paths** instead of absolute paths to work properly... otherwise they will not found the tracked folder as soon as the drive letter changes. e.g, .\profile\playlist_manager\server\

- **Relative paths for files always are checked against foobar path:** this is true only at places like the properties panel, etc.[78] i.e. '.\profile\playlist_manager\server\' equals to 'D:\foobar2000\profile\playlist_manager\server\'.

- **Relative paths on playlists should be preferred:** this is specially a must when the music is stored along the disk Foobar2000 resides in. Otherwise the files would not be found as soon as the drive letter changes (see previous tip). This may greatly affect the speed of the loading playlist process or even make it fail.

- Use the **'Check playlists consistency...'** (right button menu) to ensure all is properly set (library, configuration, playlists, ...) **on portable installations**; also handy to ensure all playlists items are found and within the library.

- Once the panel is set properly, it just takes a matter of seconds to copy the entire manager 'as is' to other installations: the config files (FOOBAR_PROFILE_PATH\js_data*.json), properties panel (can be saved as json) and playlists folder (along its files) can be transferred without changes.

42 Sharing playlists on same network

- A typical use-case would be having **2 PCs on the same network, with one being mostly used to edit/add/sort/tag new music and the other for playback. In this situation, the aim is syncing the same playlists between the 2 instances.** Usually the library and media is on a network disk accessible by the 2 instances.

- The tracked folder can be considered as a 'playlist library', it's therefore **recommended to add the playlists in a folder placed along the media library (on same disk), and add it as a tracked folder in both Foobar2000 instances**. That's all, any playlist created/edited in any instance will be immediately synced on the other.

- Since playlists point to tracks by their paths, **it would be advisable to either set the same disk letter for the network disk in both PCs or use relative paths for the playlists** (now having the playlists in the same disk than the media comes handy).

- Finally, if you want both, having the playlists available on both PCs but **also the same playlists loaded within the UI, use the Automatic Playlist actions feature [11] to automatically load specific playlists into the UI** (or all of them).

[78] When checking tracks, their root is considered to be the playlist path.

43 Sharing playlists between different devices

- **Playlists -as pointers to files- may be shared between multiple devices, but the referenced media must be shared along them to work as intended**. See 'Export and convert' tips [44] [45] for that.

- As an alternative, **query-based playlists may be used which point to media by metadata instead of only paths/links**. For that purpose **single Auto-Playlists may be easily shared [13.2]** between multiple foobar2000's devices and users: the output will vary from one library to another and -in general- it's limited to generic tags like genre, style, etc.

- Expanding Auto-Playlist compatibility, **XBMC Smart Playlists (.xsp format) may be used to translate most Auto-Playlist queries to a format readable by Kodi-like systems**. They also allow playlist nesting (using other playlists as sources); combine that with cloning [13.3] and that's an easy way to create standard playlists based on queries and joining multiple playlists.

- For a more **track-specific approach consider using .xspf format** [32] which allows to point to specific tracks by tags like title, album or artist. Any library with tracks matching those tags would be able to load the playlist. This can be used to easily share **path agnostic playlists** on the net (like 'Top 100 Rock track') which can be loaded by any Playlist Manager instance with matched client-side tracks. Contrary to Auto-Playlists the output will be limited to the playlist size and duplicates are not allowed. Also the format is universally supported by other players, specially on Unix based OSes and software.

44 Export and convert playlists to Foobar2000 mobile

- **Foobar2000 mobile app is capable of reading .pls, .m3u and .m3u8 playlists**, in other words playlists created within the pc software can be used on the mobile app too.
- **Foobar2000 mobile app uses plain text playlists with relative paths pointing to the musical files within the device**: so the playlist file must be converted along the tracks [8.3].
- **The manager is capable of exporting the tracks along the playlist file and automatically convert the paths to relative paths**: that means that playlists are ready to be used as is using the existing exporting tools [14]: export and copy or export and convert. When the files have been exported, just copy them to the tracked folder in your device.
- **Note the playlist is set with relative paths using as root the same folder where the playlist resides, but Foobar2000 mobile can be set to store playlists in another folder different to the music folder**: in fact the same that can be seen in the manager [8.3]. You have 2 options: either setting the playlist folder as the music root folder within the app **(A)** or manually editing the exporting preset for the new root **(B)**. Note the latter is pretty easy since the manager default preset TF is set with '.\' at the beginning, so you can simple replace that part instances with '..\Music\'.

```
[SD CARD:] (A)
└─ [Music]
   ├─ My playlist.m3u8
   ├─ [Big Retro Hits 90s]
   │  ├─ 007. Chateau Pop - Maniac.mp3
   │  └─ ...
   └─ ...
```

```
[SD CARD:] (B)
├─ [Playlist]
│  └─ My playlist.m3u8
└─ [Music]
   ├─ [Big Retro Hits 90s]
   │  ├─ 007. Chateau Pop - Maniac.mp3
   │  └─ ...
   └─ ...
```

Relative path on current root (SD CARD:\\Music\):
.\Big Retro Hits 90s\007. Chateau Pop - Maniac.mp3

Relative path one level up from root (SD CARD:\\Playlists\):
..\Music\Big Retro Hits 90s\007. Chateau Pop - Maniac.mp3

- **Note the music files may be placed directly in the music folder o within subfolders**: this can be set with converter preset and the TF expression for the filename. Since both the converter preset and the manager exporting preset must use the same filename expression, the paths will be reflected in the files and the playlist.

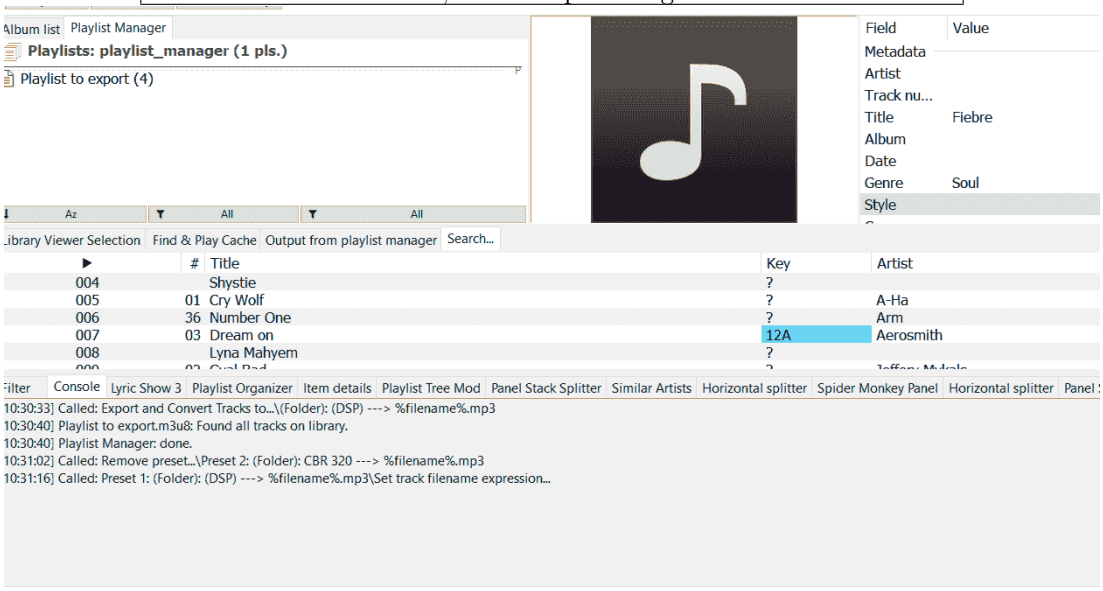
```
[SD CARD:]
└─ [Music]
   ├─ My playlist.m3u8
   ├─ 007. Chateau Pop - Maniac.mp3
   └─ ...
```

```
[SD CARD:]
└─ [Music]
   ├─ My playlist.m3u8
   ├─ [Big Retro Hits 90s]
   │  ├─ 007. Chateau Pop - Maniac.mp3
   │  └─ ...
   └─ ...
```

- **There are 2 predefined presets in the manager to simulate these steps**: they can be used 'as is' but remember to set the converter preset to use the same TF expressions!

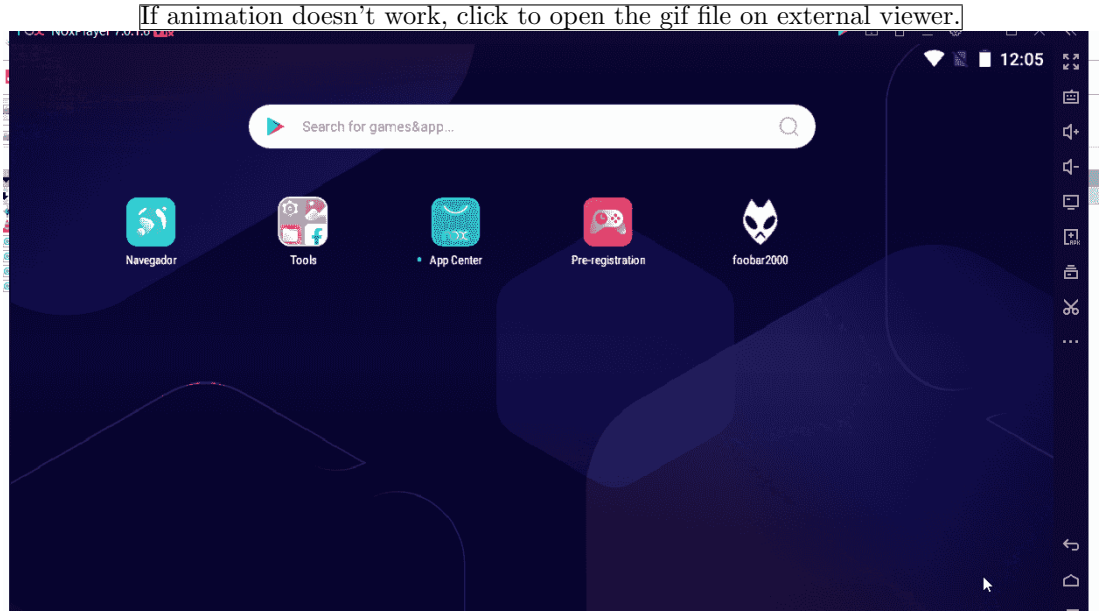
- **The following gif shows the process of exporting a playlist for another Foobar2000 mobile device** (or any software capable of reading playlists with relative paths). Note how the converter preset and manager preset filename TF expressions match (path editing may be omitted when setting properly a preset). The output folder is set to be asked at both places:

If animation doesn't work, click to open the gif file on external viewer.



- **The following gif shows the process of copying the files, with different folders for playlist and tracks, and setting Foobar2000 mobile to track them properly.** Done on an Android emulator, the steps are exactly the same on physical devices:

If animation doesn't work, click to open the gif file on external viewer.



45 Export and convert playlists to Kodi, Librelec, ...

- **Kodi based systems are capable of reading .m3u, .m3u8 and .xsp playlists**, so playlists created by the manager can be used too.

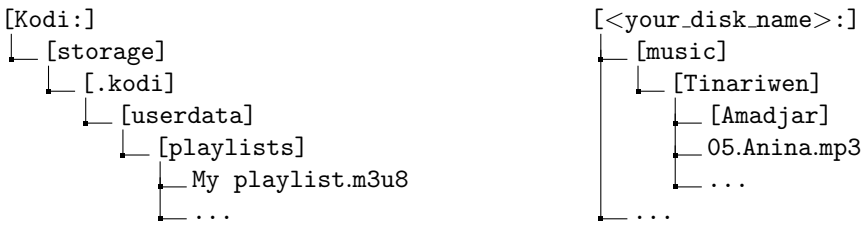
- **[By default] Kodi associates .m3u8 playlists to video streams so it's recommended to use .m3u format for music-only playlists instead**: note this is only a matter of changing their extension, since both formats are equivalent (and created the same by the manager). [29].

- **Playlists must be stored in a special folder at the system drive:** it varies according to the OS. Obviously playlists may be loaded as files from any location (using the file explorer), but adding them to the special folder will make them available on the corresponding media section (music).

OS	Playlist Folder
Android:	Android/data/org.xbmc.kodi/files/.kodi/userdata/playlists
iOS(s)	/private/var/mobile/Library/Preferences/Kodi/userdata/playlists
LibreELEC	/storage/.kodi/userdata/playlists
Linux	/.kodi/userdata/playlists
Mac	/Users/<your_user_name>/Library/Application Support/Kodi/userdata/playlists
Windows	%APPDATA%\Kodi\userdata\playlists
	...

Table 21: Special playlists folder (on userdata folder).

- **Since playlists are on the system drive and is usually placed at a different disk, absolute paths may need to be used:** but the path can not be the same than the one found on Foobar2000's pc. If your music on the PC can be found at 'H:\music\', on Kodi it will be something like '\media\<your_disk_name>\music\'[79]. In other words, the playlists need different absolute paths added in prevision to where the files will be stored on the Kodi system.



- **There are some presets on the manager ready to be used for Kodi:** they are set to convert the output playlists to .m3u (no matter the original playlist format) and the paths according to the previous point (where the external disk is mounted). **'<your_disk_name>' must be edited on configuration menu ('track filename expression') before using**. Once done, the playlists are ready to be used 'as is' and can be copied to the special playlists folder. The music, obviously, must be present on the external disk at the appropriate path (usually <your_disk_name>:\music\).

[79] On unix-like systems at least (and that includes android), drives are mounted to folders. So you have to find the path to that mounted folder to point to the external drive. The path varies from OS to OS but the idea remains the same.

- **There is an alternative method to using specific absolute or relative paths on Kodi: Path substitution**. This is and avanced method which involves telling Kodi to translate an specific path (whenever its found on a file) to another one. To use it, create a file named 'advancedsettings.xml' in your userdata folder (look at the special playlists folder) like this:

```
<advancedsettings>
    <pathsubstitution>
        <!– 'H:/music' to 'ExtDisk:/music'. Done on real time by Kodi–>
        <substitute>
            <from>H:/music/</from>
            <to>/media/ExtDisk/music/</to>
        </substitute>
    </pathsubstitution>
</advancedsettings>
```

From Foobar 2000's system

```
[D:]
└── [Software]
    └── [foobar2000]
        └── [profile]
            └── [playlist_manager]
                ├── My playlist.m3u8
                └── ...
```

To Kody system:

```
[Kodi:]
└── [storage]
    └── [.kodi]
        └── [userdata]
            └── [playlists]
                ├── My playlist.m3u8
                └── ...
```

→

```
[H:]
└── [music]
    ├── [Tinariwen]
    │   └── [Amadjar]
    │       ├── 05.Anina.flac
    │       └── ...
    └── ...
```

```
[ExtDisk:]
└── [music]
    ├── [Tinariwen]
    │   └── [Amadjar]
    │       ├── 05.Anina.mp3
    │       └── ...
    └── ...
```

Note on this example the external disk is in fact mounted on the Kodi drive too, also the kodi external disk has tracks converted to mp3:

```
[Kodi:]
├── [storage]
│   └── ...
├── [media]
│   └── [ExtDisk]
│       └── [music]
│           └── [Tinariwen]
│               └── [Amadjar]
│                   ├── 05.Anina.mp3
│                   └── ...
└── ...
```

46 Tag automation

- **Tracks are never tagged twice using the Track tags feature**, so there is no need to check if any of them has been already tagged or not before.
- **Using Auto-playlists along the Track tags feature, items on library can be auto-tagged on startup** according to some conditions without any user input. For ex. tagging all tracks with 'Rock' and 'Acoustic' as genre and BPM lower than 90 with an specific mood or occasion tag. Just set it and forget, it will be done on every new track added on library as soon as it matches the conditions.
- **Using standard playlist along the Track tags feature is an easy way to manually tag tracks on batches** while listening to them. For ex. to add tags like 'Instrumental' or 'Acoustic', 2 playlists may be used for auto-tagging with these conditions and just listen to the music; as soon as one track must be tagged it takes a second to send the current track to any of the 2 playlist (Shift + L. Click) to tag it.
- **Use category filtering to have a virtual folder of 'tagging playlists' which would only be shown when needed**, thus not convoluting the UI the rest of the time. This has some improvements and limits compared to the use of custom buttons and Mass-tagger presets.

47 Pools using Auto-playlists and tags

- **Native Foobar2000 limits the sourcing of Auto-playlists to the library, not allowing to create playlists from playlists**... although this can be simulated copying the original query and using in the new playlist, it becomes easily convoluted as soon as you do it 2 or 3 times. **As an alternative, Track tags feature may be used**. If you **set any playlist to automatically tag its tracks with an specific tag**, for ex. playlist = 'Summer', you can **then create another Autoplaylist with a query for that tag** 'PLAYLIST IS Summer'.
- The same trick can be used to **merge multiple playlist sources into one** ('PLAYLIST IS Summer OR PLAYLIST IS Chill OR PLAYLIST IS BEST'), **effectively using other playlists as pools**. Note 'sources' are not limited to Auto-playlists and that's the real power of this solution, both standard playlists and Auto-playlists which automatically tag tracks this way may be used.
- Alternatively, **Playlist-Tools-SMP [VI] allows to directly use playlists as sources for pools without requiring the use of intermediate tags**.
- Used along Playlist-Tools-SMP [VI], the emulation of pools can be greatly enhanced with the Remove Duplicates or X random selection features to create playlists in a matter of seconds from your pre-selected set of tracks.

48 Pools using Playlist-Tools-SMP

- **Feature is enabled automatically when using both scripts in the same foobar2000 instance**. As previously noted, Playlist-Tools-SMP [VI] has a random pools feature which can use playlists as sources to output a random list of tracks from all its pools. Playlist-Tools-SMP checks periodically if there is a Playlist Manager panel and retrieves the list of playlist being tracked to use them as source if required. In other words: there is no difference between loaded playlists within the UI or playlist files.

- There is a **preset example at "presets\Playlist Tools\pools\test_playlisManager.json"** which uses a playlist named "test" as source (after importing it). The file may be used as reference to create your own presets.

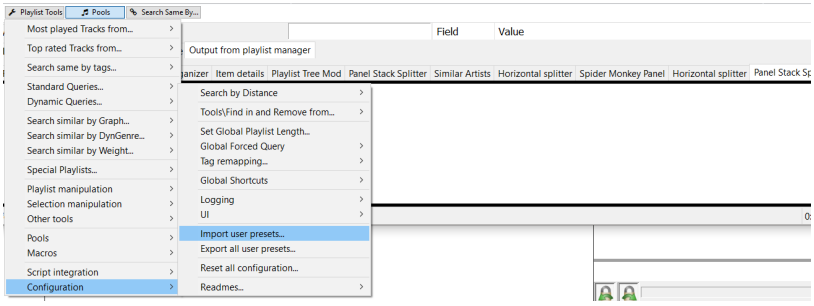


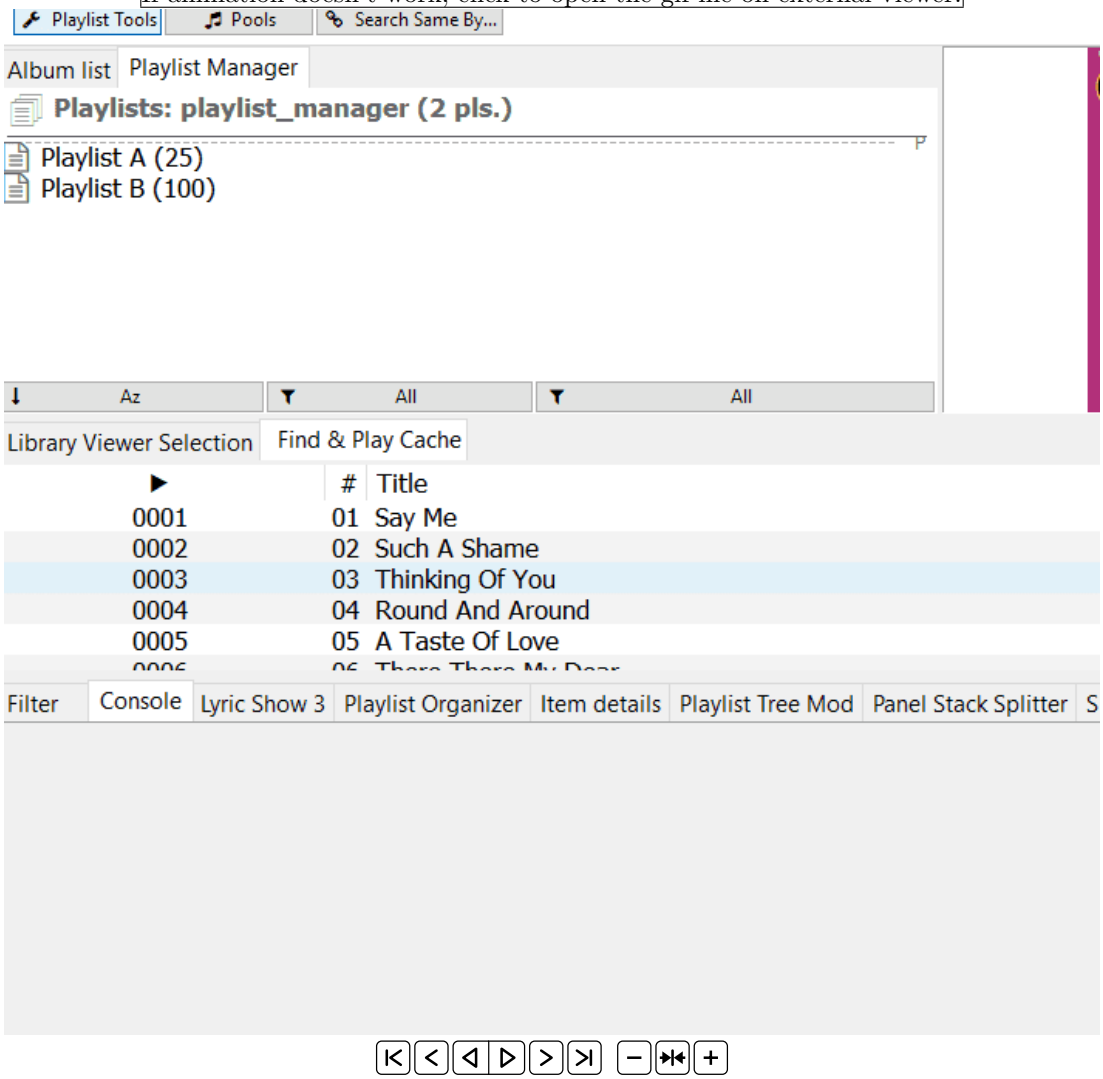
Figure 138: Importing a preset file in Playlist-Tools-SMP.

- For example **to use "Playlist A" and "Playlist B" as sources, a preset file would have to be set like this**:

```
{
    "readme": "Playlist A and Playlist B as sources, 25 tracks per playlist.",
    "pools": [{
        "name": "Playlist Manager test V2",
        "pool": {
            "fromPls": {
                "Playlist A": 25,
                "Playlist B": 25
            },
            "query": {
                "Playlist A": "ALL",
                "Playlist B": "ALL"
            },
            "toPls": "Output playlist",
            "sort": "%playlist_index%",
            "pickMethod": {
                "Playlist A": "random",
                "Playlist B": "random"
            }
        }
    }]
}
```

- Alternatively, a **custom pool may be directly executed once via menus -or added as a new entry for later use-**:

If animation doesn't work, click to open the gif file on external viewer.



- **Playlists tracked by the manager(s) may be used along other sources** like playlists within the UI, library (+query) and other complex functions:

```
[...]
"fromPls": {
    "Playlist A": 25,
    "_LIBRARY_0": 7,
    "_SEARCHBYGRAPH_0": 13
},
```

- Playlist follow this rule when **trying to find a match: loaded playlist first, then matching playlists by name (metadata) and finally by filename**.

49 Smart Playlists queries

- As specified by the .xsp format [35], **XBMC's queries can be translated into Foobar2000 ones without any loss of information**, i.e. the translation is 100% equivalent. The following table provides a guide:

XBMC's rule	Foobar2000's Query
contains	FIELD HAS X
doesnotcontain	NOT (FIELD HAS X)
is	FIELD IS X
isnot	NOT (FIELD IS X)
startswith	$strstr(%FIELD%,X) EQUAL 1
endswith	$strstr($right(%artist%,$len(X)),X) EQUAL 1
lessthan	FIELD LESS X
greaterthan	FIELD GREATER X
after	DATEFIELD AFTER X
before	DATEFIELD BEFORE X
inthelast	DATEFIELD DURING LAST X
notinthelast	NOT (DATEFIELD DURING LAST X)

Table 22: XBMC Query translation.

- **Whenever a rule has multiple values (X), the translated query must be expanded using 'OR' as nexus**.

<rule field="genre" operator="is"> <value>Rock</value> <value>Folk</value> </rule>	(GENRE IS Rock OR GENRE IS Folk)

- **In the case of the rule being a negative** (with 'NOT' at the beginning), note **the expression can also be expanded with '[AND] NOT'** as nexus.

<rule field="genre" operator="isnot"> <value>Rock</value> <value>Folk</value> </rule>	NOT (GENRE IS Rock OR GENRE IS Folk) (NOT GENRE IS Rock AND NOT GENRE IS Folk)

- The last point implies that while **XBMC←Foobar2000 translation is always possible, the transformation is not injective** (same XBMC query has multiple Foobar2000 results). The transformation is also not surjective, **not all Foobar2000 queries have an XBMC equivalence (most TF expressions are not supported)**. As consequence **XBMC→Foobar2000 transformation is not always possible**, and even worse, there is an arbitrary number of different ways to write the same query (same output). Not only because negative rules may be written as 'NOT (A IS x OR A IS y ...)' and also as '(NOT A IS x AND NOT A IS y ...)', but they may also be written as '(NOT A IS x) AND (NOT A Is y) ...', etc. Foobar2000 queries are open to be written on any desired order with arbitrary parentheses.

- All these characteristics make **XBMC→Foobar2000 query automatic transformation not 100% accurate (even when such transformation is possible), relying on some form of heuristics and "known patterns". This implies there may be some cases where the Foobar2000 query is not properly recognized as an XBMC pattern, needing some refactoring or rewriting to be recognized.**

XBMC's rule	XBMC's match	Foobar2000 Query
is(Rock), is(Folk)	one (OR)	GENRE IS Rock OR GENRE IS Folk
is([Rock,Folk])	all (AND)	(GENRE IS Rock OR GENRE IS Folk)
is(Rock), is(Blues), is(Folk)	one (OR)	GENRE IS Rock OR STYLE IS Blues OR GENRE IS Folk
is(Instrumental), is([Rock,Folk])	all (AND)	GENRE IS Instrumental AND (GENRE IS Rock OR GENRE IS Folk)
Does not have a translation!	-	GENRE IS Instrumental AND (GENRE IS Rock OR STYLE IS Blues OR GENRE IS Folk)
Does not have a translation!	-	(GENRE IS Instrumental) AND (GENRE IS Rock OR GENRE IS Folk) OR (STYLE IS Blues)
is(Instrumental), is([Rock,Folk]), is([Blues,Electric])	all (AND)	(GENRE IS Instrumental) AND (GENRE IS Rock OR GENRE IS Folk) AND (STYLE IS Blues OR STYLE IS Electric)

Table 23: XBMC←Foobar2000 Query translation examples.

- As can be seen on 2 of the last 3 examples (which are equivalent ways of writing the same query), **it's not possible to have in the same rule global 'AND' and 'OR' statements since that's a limit of XBMC matching (which imposes to match either all rules -AND- or one -OR-)**. The heuristics try to group patterns with 'OR' within a parentheses though, because they are not global.

- Note in any case, **even if not all statements are matched, a translated rule is output to try to offer the most similar rule to the query input**:

XBMC's rule	XBMC's match	Foobar2000 Query
is(Instrumental)	all (AND)	GENRE IS Instrumental AND (GENRE IS Rock OR STYLE IS Blues OR GENRE IS Folk)
is(Instrumental), is([Rock,Folk])	all (AND)	(GENRE IS Instrumental) AND (GENRE IS Rock OR GENRE IS Folk) OR (STYLE IS Blues)

Table 24: XBMC←Foobar2000 translation error output.

- Since globally all or one of the rules must be matched, **the only way to create more complex queries involves multiple Smart Playlists. Playlists may query other playlists**, thus allowing nesting conditions in an indirect way. To create a Smart Playlist with '(GENRE IS A AND STYLE IS B) OR (GENRE IS C AND STYLE IS D)', 3 Smart Playlists are needed:

  First Playlist: '(GENRE IS A) AND (STYLE IS B)'
  Second Playlist: '(GENRE IS C) AND (STYLE IS D)'

  ⇓⇓⇓

  Third Playlist: '#PLAYLIST# IS First OR #PLAYLIST# IS Second'

50 Pools using Smart Playlists

- As seen on [49], **Smart Playlists may use other playlists as sources to nest conditions. But that's is not limited to using other Smart Playlists as sources, but any playlist** in fact: **Auto-Playlists (at least within Foobar2000 context), standard playlists**, etc. In other words, Smart Playlists allow to use any source as pool.

- **One advantage over using Auto-Playlists and tags [47] for the same purpose is that tagging is not required at all in this case, so the original files may be left untouched**.

- **Another advantage is limiting the number of tracks at output**, which may be set on Smart Playlists but not on Auto-Playlists... so it may make more sense when trying to create 'pools'.

- **Limit is applied after sorting**, so once an Smart Playlist has loaded all the tracks from its sources, it sorts them and then takes only the first X elements.

- **Limit and sorting is applied recursively. i.e. any source called by the Smart Playlist may also have sorting and/or limit rules**, which are applied before sending the track list to the caller. This may be used to artificially apply limits to Auto-Playlists: you may have an Auto-Playlist (A) used as source by an Smart-Playlist (B) which has a limit set. Effectively that's like calling A and only getting X tracks. If sort is random, then you are getting X different random tracks from an Auto-playlist every time it's executed.

- **Since Auto-Playlists allow more complex queries than Smart Playlist, this type of nesting may be used as a way to simply impose limits to them**. In the previous example, a third Smart Playlist (C) could use (B) as source too, along other playlists. (B) would be only an intermediary to impose how many tracks are taken from a pool.

- **Note Playlist-Tools-SMP [48] still allows a greater degree of customization**: different picking methods, allowing to use complex similarity functions as sources, etc. **But in a playlist context, they are pretty much equivalent** using the nesting method (although clearly some things may require a really high number of intermediate Smart Playlists).

- Finally, it should be noted that **nesting playlist is also allowed on Kodi/XBMC systems, so they will work the same on Foobar2000 or other devices**. The only difference on use is that Auto-Playlists doesn't exist on other devices, so they must be obviously be substituted (when used as sources) by other Smart Playlists or standard ones.

51 Working with locked playlists

- **Forget about lock status on Foobar2000 playlists.** There are some plugins or even Spider Monkey Panel scripts which allow to lock/unlock native playlists in some way or another. While it may come useful for advanced users, regular users should simply stick to the manager lock features. Playlists now have a physical file counterpart which can be locked, so there is no need to lock playlists within UI for changes[80].

- **Instead of forbidding edits, just reload the playlist.** The native approach focuses on forbidding specific actions on playlists (reordering, adding, removing items, etc.). Locking the playlist file allows any of those edits on the loaded playlist and if you want to revert them just undo them or reload the playlist to discard all of them and revert it to the original version.

- **Edits on locked playlists can be saved if forced to do so but never automatically.** By default, any change made to a locked playlist loaded within Foobar2000 is not auto-saved and therefore only temporarily stored while the playlist remains opened... but this can be bypassed on demand without unlocking the playlist using the 'Force playlist file update' entry on the selected playlist contextual menu [18].

52 Autosave & Autobackup component

- **If using Autosave & Autobackup component it's recommended to add 'js_data' and 'playlist_manager' folders** at 'Preferences\Advanced\Autosave & Autobackup\Files and directories to backup (...)'. 'js_data' contains config json data for most of these scripts, while 'playlist_manager' folder is the default folder to save playlists filesr.

- When the default path is changed or when using multiple playlist manager panels (and thus different paths), don't forget to add them to Autosave config too. There is a limit though, **paths outside the profile folder can not be autosaved by the component. That happens when tracking a playlist folder at the place where the music resides (for relative paths playlist usage), network locations, etc.**

- In some cases this **limitation can be bypassed by creating a SymLink at the profile folder to the external path**. For ex:
'C:\Foobar2000\profile\my_external_folder\' → symlink → 'H:\Music\'

[80] This also simplifies some quirks about playlist locks which involve the type of lock and owners...

53 Customizable icons for categories

- **There is an additional hidden feature which allows to customize the icon on the header according to the shown category:** to enable it, a file must be created along the manager's (auto)playlists json file [2.1]. With '_config' appended to the filename.

```
[C:/Users/XXX/AppData/Roaming]
└─ [foobar2000]
   └─ [js_data]
      ├─ playlistManager_playlists.json
      ├─ playlistManager_playlists_config.json
      └─ ...
```

- **The file is a simple json file, whose keys within the 'category' section must be the categories to be matched**; the values are the **Unicode chars of the icon** (\f000, \f01c, ...). Font Awesome is used, a cheat-sheet can be found here: .https://www.fontawesomecheatsheet.com/

- **An example file can be found below:**

```
{
    "ui": {
        "icons": {
            "category": {
                "Favorites": "\uf004",
                "Tagging": "\uf02c",
                "Pools": "\uf0e8",
                "Party": "\uf000",
                "Share": "\uf045"
            }
        }
    }
}
```

- **Icons set on the file will be used when panel gets manually reloaded, on manual refresh [17] or on next startup.**

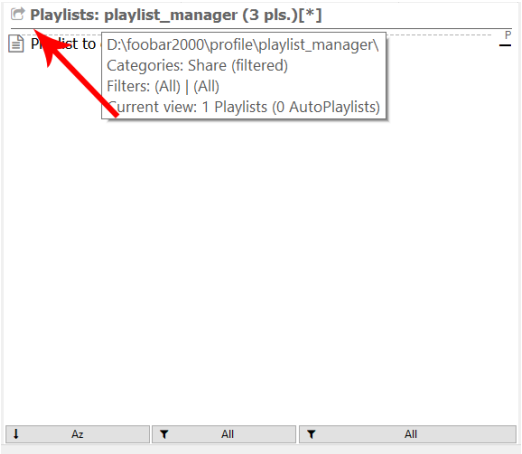


Figure 139: 'Share' category.

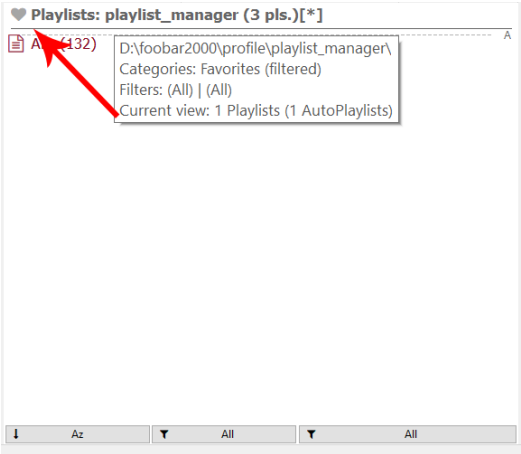


Figure 140: 'Favorites' category.

54 Recycle Bin on network drives

- **Playlist files can not be restored on network drives.** Network drives don't have Recycle Bin by default, so files are permanently deleted without possible restoration. This affects to deleted playlist files before updating, backups, etc.

- As a workaround, **the Recycle Bin may be configured to include additional folders/drives editing the registry.**

- **Check the following folder for instructions:** '.\presets\Network'

  + **A**dd the corresponding registry entries according to your system (32 or 64 bits SO).

  + **Restart the system** to apply the changes.

  + **Repeat the process for every network drive which will be used.** Adding a different GUID for each one. Don't forget to create a clean entries file for each GUID!

  + **C**hanges may be always reverted by deleting the entries associated t such GUID's.

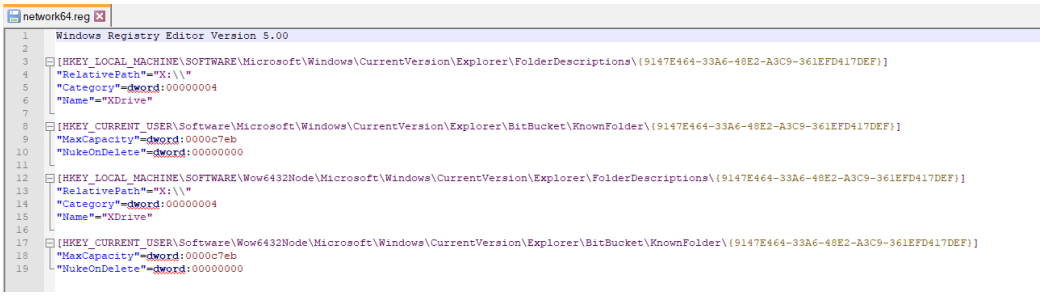


Figure 141: Example of registry entries for x64 SO and a drive at 'X:\'

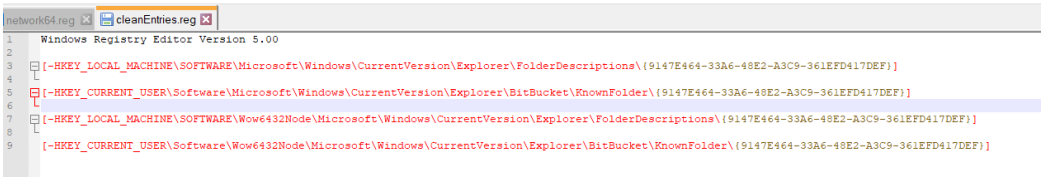


Figure 142: Deleting the entries for the GUID reverts the changes.

- **More info can be found at**: social.technet.microsoft

- **No further support will be given related to this** (not an script related topic).

Part X

Technical notes

- **All json files used by the scripts are read as UTF-8 (whether they have BOM or not)**, being the only **exception those imported from marc2003's panel** [13.1] according to the popup option chosen. In such case code-page detection relies in SMP heuristics (comments on FAQ apply [VIII]). Have this in mind when editing or creating json files with external editors.

- **Any text file created by the scripts will be UTF-8 encoded without BOM**, by default. **BOM can only be enabled for playlist files**, if desired, using the configuration menus [8.4].

- **Writing BOM, for playlist files, is disabled by default** since it's a windows-only idiotic thing which breaks compatibility with many programs (written to properly identify UTF-8 files).

- By this order, .**m3u8 playlist files are always read as UTF-8**. .xsp and .xspf files rely on the mandatory **XML encoding flag** [32][35]. Then .m3u files rely on the **encoding flag** [29] if present. *In any other case code-page detection relies on SMP heuristics* (comments on FAQ apply [VIII]). Have this in mind when using playlist files from external software.

- **Locking of .fpl playlists by default can only be disabled using the properties panel** to ensure regular users don't mess with it. Any other configuration related item can be set via menus.

- **Number of zip backups allowed can be set using the properties panel**. Default is 50.

- **Library items' paths are cached to provide fast playlist file loading, but cache must be rebuilt every time the library changes (paths or adding/removing tracks)**. Errors on the cache may corrupt playlists with wrong tracks being saved to them. To avoid this possibility safechecks have been added to ensure the original path and the cached item's path are the same, throwing a warning whenever they don't match. Cache is also automatically rebuilt on library changes, so for regular users all is done automatically under the hood. Note that during a cache rebuilding, playlist loading may fail throwing the aforementioned warning; this is expected and it will use native Foobar2000 playlist loading as fallback (when forced). To avoid such situation and make it easier to know when this happens, panel is disabled during the process showing a loading animation with a tooltip.

- **Multiple Playlist Manager panels share the same cache**: there is a built-in instance manager which ensures that anytime there are multiple Playlist Manager panels within the same Foobar2000 instance, the cache is calculated only once by one panel and shared to the rest (instead of being calculated in parallel).

Part XI

Known problems

- **Spider Monkey Panel may crash -without known reason- at startup when trying to install the manager. The error will warn about script files not being found, although they are in place.** This is due to incorrect relative path handling on some systems. In such case, **take a look at the '_switchPaths.zip', decompress it and follow its instructions**. Scripts will be automatically edited to use absolute paths at script loading, thus fixing the problem. The cmd file (.bat) will have to be rerun on future playlist manager updates. Obviously this is only a workaround and a proper fix is expected on posterior SMP releases.
- **Spider Monkey Panel may show a warning popup about a script being unresponsive.** This is not an script error but an SMP feature to avoid scripts blocking the entire software (please don't report it). It's common for processor intensive tasks which require more time to finish (just press continue). Clicking 'stop the script' will crash the panel.

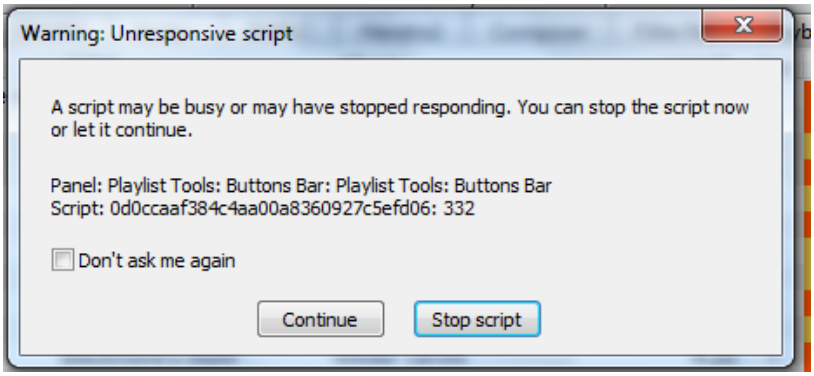


Figure 143: Slow script warning popup. Click on continue.

Since some operations may need more time than usual, to avoid seeing the popup on consecutive sessions, it's recommended to change the time limit at configuration.

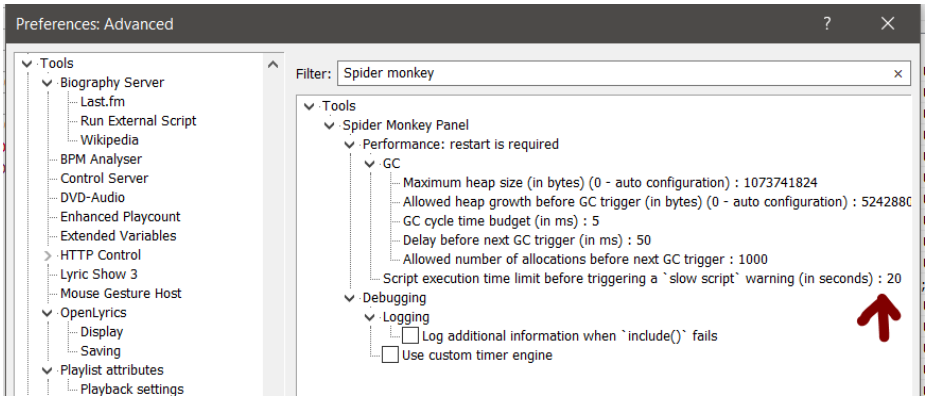


Figure 144: SMP Advanced configuration. Set it to a high value until the popup is not shown.

- **Spider Monkey Panel (any version) may incorrectly identify code-page while reading text files**, thus reporting wrong values at some instances. This usually happens while reading playlist files or importing tracks lists, with track names or artists using exotic chars (ä,â, ...) getting corrupted. Multiple code checks have been added to minimize these situations in any case. See FAQ [VIII].

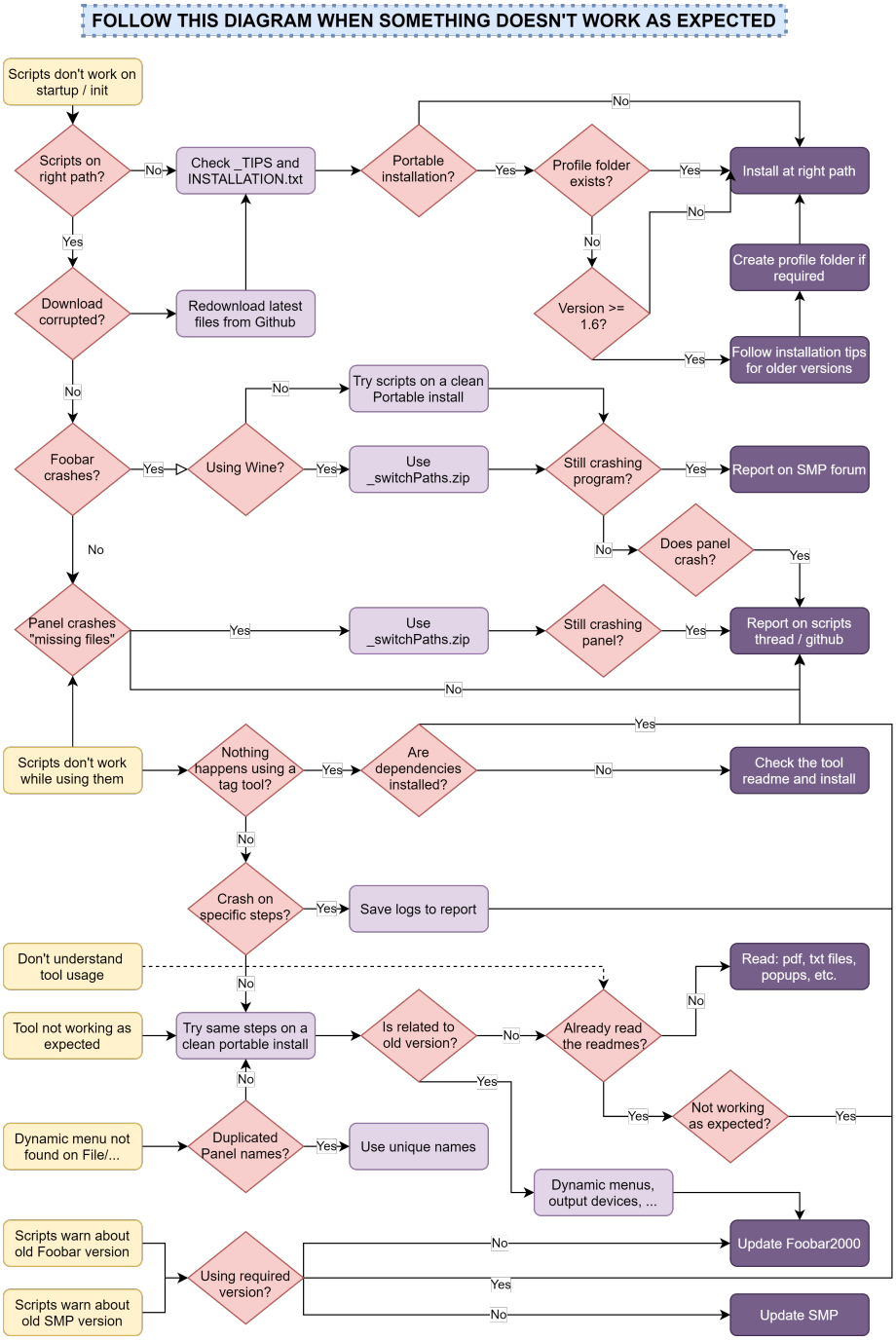
- **The instance manager will fail tracking panels if a panel crashes**, thus after reloading a panel, it will consider there is only one instance of Playlist Manager even if there are multiple ones. A restart of Foobar2000 is required to fix it or reloading all other Playlist Manager panels. Note this is only a temporal problem (until next startup) and it only affects cache calculation in any case (if it needs to be re-calculated, it will be done on all the panels in parallel).

- **Playlist files can not be restored on network drives.** Network drives don't have Recycle Bin by default, so files are permanently deleted without possible restoration. This affects to deleted playlist files before updating, backups, etc. As a workaround, the Recycle Bin may be configured to include additional folders/drives [54]. No further support will be given related to this (not an script related topic).

- **Auto-backup [10.3] (7za) is not working as expected on some OS's.** CMD tools may fail on Unix systems when using Wine prefixes. In particular, for 7za, since the binaries are run via command line using bat files, OS Architecture must be matched (x64, ...). Some measures have been taken to account for it automatically, since the 64 bit binaries are preferred for performance. No further support will be given (although it's possible to replace the 64 bit binaries with 32 bits versions).

Part XII

Bug procedure



Part XIII

License

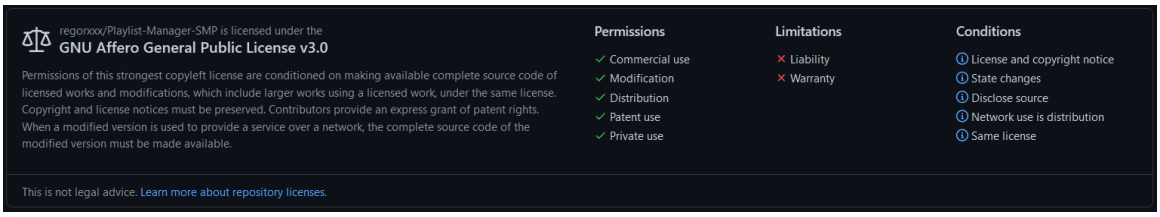


Figure 145: License abstract.

GNU AFFERO GENERAL PUBLIC LICENSE
Version 3, 19 November 2007
Copyright (C) 2007 Free Software Foundation, Inc. https://fsf.org/

Everyone is permitted to copy and distribute verbatim copies of this license document, but changing it is not allowed.

Preamble

The GNU Affero General Public License is a free, copyleft license for software and other kinds of works, specifically designed to ensure cooperation with the community in the case of network server software.

The licenses for most software and other practical works are designed to take away your freedom to share and change the works. By contrast, our General Public Licenses are intended to guarantee your freedom to share and change all versions of a program–to make sure it remains free software for all its users.

When we speak of free software, we are referring to freedom, not price. Our General Public Licenses are designed to make sure that you have the freedom to distribute copies of free software (and charge for them if you wish), that you receive source code or can get it if you want it, that you can change the software or use pieces of it in new free programs, and that you know you can do these things.

Developers that use our General Public Licenses protect your rights with two steps: (1) assert copyright on the software, and (2) offer you this License which gives you legal permission to copy, distribute and/or modify the software.

A secondary benefit of defending all users' freedom is that improvements made in alternate versions of the program, if they receive widespread use, become available for other developers to incorporate. Many developers of free software are heartened and encouraged by the resulting cooperation. However, in the case of software used on network servers, this result may fail to come about. The GNU General Public License permits making a modified version and letting the public access it on a server without ever releasing its source code to the public.

The GNU Affero General Public License is designed specifically to ensure that, in such cases, the modified source code becomes available to the community. It requires the operator of a network server to provide the source code of the modified version running there to the users of that server. Therefore, public use of a modified version, on a publicly accessible server, gives the public access to the source code of the modified version.

An older license, called the Affero General Public License and published by Affero, was designed to accomplish similar goals. This is different license, not a version of the Affero GPL, but Affero has released a new version of the Affero GPL which permits relicensing under this license.

The precise terms and conditions for copying, distribution and modification follow.

TERMS AND CONDITIONS

0. Definitions.

"This License" refers to version 3 of the GNU Affero General Public License.

"Copyright" also means copyright-like laws that apply to other kinds of works, such as semiconductor masks.

"The Program" refers to any copyrightable work licensed under this License. Each licensee is addressed as "you". "Licensees" and "recipients" may be individuals or organizations.

To "modify" a work means to copy from or adapt all or part of the work in a fashion requiring copyright permission, other than the making of an exact copy. The resulting work is called a "modified version" of the earlier work or a work "based on" the earlier work.

A "covered work" means either the unmodified Program or a work based on the Program.

To "propagate" a work means to do anything with it that, without permission, would make you directly or secondarily liable for infringement under applicable copyright law, except executing it on a computer or modifying a private copy. Propagation includes copying, distribution (with or without modification), making available to the public, and in some countries other activities as well.

To "convey" a work means any kind of propagation that enables other parties to make or receive copies. Mere interaction with a user through a computer network, with no transfer of a copy, is not conveying.

An interactive user interface displays "Appropriate Legal Notices" to the extent that it includes a convenient and prominently visible feature that (1) displays an appropriate copyright notice, and (2) tells the user that there is no warranty for the work (except to the extent that warranties are provided), that licensees may convey the work under this License, and how to view a copy of this License. If the interface presents a list of user commands or options, such as a menu, a prominent item in the list meets this criterion.

1. Source Code.

The "source code" for a work means the preferred form of the work for making modifications to it. "Object code" means any non-source form of a work.

A "Standard Interface" means an interface that either is an official standard defined by a recognized standards body, or, in the case of interfaces specified for a particular programming

language, one that is widely used among developers working in that language.

The "System Libraries" of an executable work include anything, other than the work as a whole, that (a) is included in the normal form of packaging a Major Component, but which is not part of that Major Component, and (b) serves only to enable use of the work with that Major Component, or to implement a Standard Interface for which an implementation is available to the public in source code form. A "Major Component", in this context, means a major essential component (kernel, window system, and so on) of the specific operating system (if any) on which the executable work runs, or a compiler used to produce the work, or an object code interpreter used to run it.

The "Corresponding Source" for a work in object code form means all the source code needed to generate, install, and (for an executable work) run the object code and to modify the work, including scripts to control those activities. However, it does not include the work's System Libraries, or general-purpose tools or generally available free programs which are used unmodified in performing those activities but which are not part of the work. For example, Corresponding Source includes interface definition files associated with source files for the work, and the source code for shared libraries and dynamically linked subprograms that the work is specifically designed to require, such as by intimate data communication or control flow between those subprograms and other parts of the work.

The Corresponding Source need not include anything that users can regenerate automatically from other parts of the Corresponding Source.

The Corresponding Source for a work in source code form is that same work.

2. Basic Permissions.

All rights granted under this License are granted for the term of copyright on the Program, and are irrevocable provided the stated conditions are met. This License explicitly affirms your unlimited permission to run the unmodified Program. The output from running a covered work is covered by this License only if the output, given its content, constitutes a covered work. This License acknowledges your rights of fair use or other equivalent, as provided by copyright law.

You may make, run and propagate covered works that you do not convey, without conditions so long as your license otherwise remains in force. You may convey covered works to others for the sole purpose of having them make modifications exclusively for you, or provide you with facilities for running those works, provided that you comply with the terms of this License in conveying all material for which you do not control copyright. Those thus making or running the covered works for you must do so exclusively on your behalf, under your direction and control, on terms that prohibit them from making any copies of your copyrighted material outside their relationship with you.

Conveying under any other circumstances is permitted solely under the conditions stated below. Sublicensing is not allowed; section 10 makes it unnecessary.

3. Protecting Users' Legal Rights From Anti-Circumvention Law.

No covered work shall be deemed part of an effective technological measure under any applicable law fulfilling obligations under article 11 of the WIPO copyright treaty adopted on 20 December 1996, or similar laws prohibiting or restricting circumvention of such measures.

When you convey a covered work, you waive any legal power to forbid circumvention of technological measures to the extent such circumvention is effected by exercising rights under this License with respect to the covered work, and you disclaim any intention to limit operation or

modification of the work as a means of enforcing, against the work's users, your or third parties' legal rights to forbid circumvention of technological measures.

4. Conveying Verbatim Copies.

You may convey verbatim copies of the Program's source code as you receive it, in any medium, provided that you conspicuously and appropriately publish on each copy an appropriate copyright notice; keep intact all notices stating that this License and any non-permissive terms added in accord with section 7 apply to the code; keep intact all notices of the absence of any warranty; and give all recipients a copy of this License along with the Program.

You may charge any price or no price for each copy that you convey, and you may offer support or warranty protection for a fee.

5. Conveying Modified Source Versions.

You may convey a work based on the Program, or the modifications to produce it from the Program, in the form of source code under the terms of section 4, provided that you also meet all of these conditions:

a) The work must carry prominent notices stating that you modified it, and giving a relevant date.

b) The work must carry prominent notices stating that it is released under this License and any conditions added under section 7. This requirement modifies the requirement in section 4 to "keep intact all notices".

c) You must license the entire work, as a whole, under this License to anyone who comes into possession of a copy. This License will therefore apply, along with any applicable section 7 additional terms, to the whole of the work, and all its parts, regardless of how they are packaged. This License gives no permission to license the work in any other way, but it does not invalidate such permission if you have separately received it.

d) If the work has interactive user interfaces, each must display Appropriate Legal Notices; however, if the Program has interactive interfaces that do not display Appropriate Legal Notices, your work need not make them do so.

A compilation of a covered work with other separate and independent works, which are not by their nature extensions of the covered work, and which are not combined with it such as to form a larger program, in or on a volume of a storage or distribution medium, is called an "aggregate" if the compilation and its resulting copyright are not used to limit the access or legal rights of the compilation's users beyond what the individual works permit. Inclusion of a covered work in an aggregate does not cause this License to apply to the other parts of the aggregate.

6. Conveying Non-Source Forms.

You may convey a covered work in object code form under the terms of sections 4 and 5, provided that you also convey the machine-readable Corresponding Source under the terms of this License, in one of these ways:

a) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by the Corresponding Source fixed on a durable physical medium customarily used for software interchange.

b) Convey the object code in, or embodied in, a physical product (including a physical distribution medium), accompanied by a written offer, valid for at least three years and valid for as long as you offer spare parts or customer support for that product model, to give anyone who possesses the object code either (1) a copy of the Corresponding Source for all the software in the product that is covered by this License, on a durable physical medium customarily used for software interchange, for a price no more than your reasonable cost of physically performing this conveying of source, or (2) access to copy the Corresponding Source from a network server at no charge.

c) Convey individual copies of the object code with a copy of the written offer to provide the Corresponding Source. This alternative is allowed only occasionally and noncommercially, and only if you received the object code with such an offer, in accord with subsection 6b.

d) Convey the object code by offering access from a designated place (gratis or for a charge), and offer equivalent access to the Corresponding Source in the same way through the same place at no further charge. You need not require recipients to copy the Corresponding Source along with the object code. If the place to copy the object code is a network server, the Corresponding Source may be on a different server (operated by you or a third party) that supports equivalent copying facilities, provided you maintain clear directions next to the object code saying where to find the Corresponding Source. Regardless of what server hosts the Corresponding Source, you remain obligated to ensure that it is available for as long as needed to satisfy these requirements.

e) Convey the object code using peer-to-peer transmission, provided you inform other peers where the object code and Corresponding Source of the work are being offered to the general public at no charge under subsection 6d.

A separable portion of the object code, whose source code is excluded from the Corresponding Source as a System Library, need not be included in conveying the object code work.

A "User Product" is either (1) a "consumer product", which means any tangible personal property which is normally used for personal, family, or household purposes, or (2) anything designed or sold for incorporation into a dwelling. In determining whether a product is a consumer product, doubtful cases shall be resolved in favor of coverage. For a particular product received by a particular user, "normally used" refers to a typical or common use of that class of product, regardless of the status of the particular user or of the way in which the particular user actually uses, or expects or is expected to use, the product. A product is a consumer product regardless of whether the product has substantial commercial, industrial or non-consumer uses, unless such uses represent the only significant mode of use of the product.

"Installation Information" for a User Product means any methods, procedures, authorization keys, or other information required to install and execute modified versions of a covered work in that User Product from a modified version of its Corresponding Source. The information must suffice to ensure that the continued functioning of the modified object code is in no case prevented or interfered with solely because modification has been made.

If you convey an object code work under this section in, or with, or specifically for use in, a User Product, and the conveying occurs as part of a transaction in which the right of possession and use of the User Product is transferred to the recipient in perpetuity or for a fixed term (regardless of how the transaction is characterized), the Corresponding Source conveyed under this section must be accompanied by the Installation Information. But this requirement does not apply if neither you nor any third party retains the ability to install modified object code on the User Product (for example, the work has been installed in ROM).

The requirement to provide Installation Information does not include a requirement to continue to provide support service, warranty, or updates for a work that has been modified or installed by the recipient, or for the User Product in which it has been modified or installed. Access to a network may be denied when the modification itself materially and adversely affects the operation of the network or violates the rules and protocols for communication across the network.

Corresponding Source conveyed, and Installation Information provided, in accord with this section must be in a format that is publicly documented (and with an implementation available to the public in source code form), and must require no special password or key for unpacking, reading or copying.

7. Additional Terms.

"Additional permissions" are terms that supplement the terms of this License by making exceptions from one or more of its conditions. Additional permissions that are applicable to the entire Program shall be treated as though they were included in this License, to the extent that they are valid under applicable law. If additional permissions apply only to part of the Program, that part may be used separately under those permissions, but the entire Program remains governed by this License without regard to the additional permissions.

When you convey a copy of a covered work, you may at your option remove any additional permissions from that copy, or from any part of it. (Additional permissions may be written to require their own removal in certain cases when you modify the work.) You may place additional permissions on material, added by you to a covered work, for which you have or can give appropriate copyright permission.

Notwithstanding any other provision of this License, for material you add to a covered work, you may (if authorized by the copyright holders of that material) supplement the terms of this License with terms:

a) Disclaiming warranty or limiting liability differently from the terms of sections 15 and 16 of this License; or

b) Requiring preservation of specified reasonable legal notices or author attributions in that material or in the Appropriate Legal Notices displayed by works containing it; or

c) Prohibiting misrepresentation of the origin of that material, or requiring that modified versions of such material be marked in reasonable ways as different from the original version; or

d) Limiting the use for publicity purposes of names of licensors or authors of the material; or

e) Declining to grant rights under trademark law for use of some trade names, trademarks, or service marks; or

f) Requiring indemnification of licensors and authors of that material by anyone who conveys the material (or modified versions of it) with contractual assumptions of liability to the recipient, for any liability that these contractual assumptions directly impose on those licensors and authors.

All other non-permissive additional terms are considered "further restrictions" within the meaning of section 10. If the Program as you received it, or any part of it, contains a notice stating that it is governed by this License along with a term that is a further restriction, you may remove that term. If a license document contains a further restriction but permits relicensing or conveying

under this License, you may add to a covered work material governed by the terms of that license document, provided that the further restriction does not survive such relicensing or conveying.

If you add terms to a covered work in accord with this section, you must place, in the relevant source files, a statement of the additional terms that apply to those files, or a notice indicating where to find the applicable terms.

Additional terms, permissive or non-permissive, may be stated in the form of a separately written license, or stated as exceptions; the above requirements apply either way.

8. Termination.

You may not propagate or modify a covered work except as expressly provided under this License. Any attempt otherwise to propagate or modify it is void, and will automatically terminate your rights under this License (including any patent licenses granted under the third paragraph of section 11).

However, if you cease all violation of this License, then your license from a particular copyright holder is reinstated (a) provisionally, unless and until the copyright holder explicitly and finally terminates your license, and (b) permanently, if the copyright holder fails to notify you of the violation by some reasonable means prior to 60 days after the cessation.

Moreover, your license from a particular copyright holder is reinstated permanently if the copyright holder notifies you of the violation by some reasonable means, this is the first time you have received notice of violation of this License (for any work) from that copyright holder, and you cure the violation prior to 30 days after your receipt of the notice.

Termination of your rights under this section does not terminate the licenses of parties who have received copies or rights from you under this License. If your rights have been terminated and not permanently reinstated, you do not qualify to receive new licenses for the same material under section 10.

9. Acceptance Not Required for Having Copies.

You are not required to accept this License in order to receive or run a copy of the Program. Ancillary propagation of a covered work occurring solely as a consequence of using peer-to-peer transmission to receive a copy likewise does not require acceptance. However, nothing other than this License grants you permission to propagate or modify any covered work. These actions infringe copyright if you do not accept this License. Therefore, by modifying or propagating a covered work, you indicate your acceptance of this License to do so.

10. Automatic Licensing of Downstream Recipients.

Each time you convey a covered work, the recipient automatically receives a license from the original licensors, to run, modify and propagate that work, subject to this License. You are not responsible for enforcing compliance by third parties with this License.

An "entity transaction" is a transaction transferring control of an organization, or substantially all assets of one, or subdividing an organization, or merging organizations. If propagation of a covered work results from an entity transaction, each party to that transaction who receives a copy of the work also receives whatever licenses to the work the party's predecessor in interest had or could give under the previous paragraph, plus a right to possession of the Corresponding Source of the work from the predecessor in interest, if the predecessor has it or can get it with reasonable efforts.

You may not impose any further restrictions on the exercise of the rights granted or affirmed under this License. For example, you may not impose a license fee, royalty, or other charge for exercise of rights granted under this License, and you may not initiate litigation (including a cross-claim or counterclaim in a lawsuit) alleging that any patent claim is infringed by making, using, selling, offering for sale, or importing the Program or any portion of it.

11. Patents.

A "contributor" is a copyright holder who authorizes use under this License of the Program or a work on which the Program is based. The work thus licensed is called the contributor's "contributor version".

A contributor's "essential patent claims" are all patent claims owned or controlled by the contributor, whether already acquired or hereafter acquired, that would be infringed by some manner, permitted by this License, of making, using, or selling its contributor version, but do not include claims that would be infringed only as a consequence of further modification of the contributor version. For purposes of this definition, "control" includes the right to grant patent sublicenses in a manner consistent with the requirements of this License.

Each contributor grants you a non-exclusive, worldwide, royalty-free patent license under the contributor's essential patent claims, to make, use, sell, offer for sale, import and otherwise run, modify and propagate the contents of its contributor version.

In the following three paragraphs, a "patent license" is any express agreement or commitment, however denominated, not to enforce a patent (such as an express permission to practice a patent or covenant not to sue for patent infringement). To "grant" such a patent license to a party means to make such an agreement or commitment not to enforce a patent against the party.

If you convey a covered work, knowingly relying on a patent license, and the Corresponding Source of the work is not available for anyone to copy, free of charge and under the terms of this License, through a publicly available network server or other readily accessible means, then you must either (1) cause the Corresponding Source to be so available, or (2) arrange to deprive yourself of the benefit of the patent license for this particular work, or (3) arrange, in a manner consistent with the requirements of this License, to extend the patent license to downstream recipients. "Knowingly relying" means you have actual knowledge that, but for the patent license, your conveying the covered work in a country, or your recipient's use of the covered work in a country, would infringe one or more identifiable patents in that country that you have reason to believe are valid.

If, pursuant to or in connection with a single transaction or arrangement, you convey, or propagate by procuring conveyance of, a covered work, and grant a patent license to some of the parties receiving the covered work authorizing them to use, propagate, modify or convey a specific copy of the covered work, then the patent license you grant is automatically extended to all recipients of the covered work and works based on it.

A patent license is "discriminatory" if it does not include within the scope of its coverage, prohibits the exercise of, or is conditioned on the non-exercise of one or more of the rights that are specifically granted under this License. You may not convey a covered work if you are a party to an arrangement with a third party that is in the business of distributing software, under which you make payment to the third party based on the extent of your activity of conveying the work, and under which the third party grants, to any of the parties who would receive the covered work from you, a discriminatory patent license (a) in connection with copies of the covered work conveyed by you (or copies made from those copies), or (b) primarily for and in connection with specific products

or compilations that contain the covered work, unless you entered into that arrangement, or that patent license was granted, prior to 28 March 2007.

Nothing in this License shall be construed as excluding or limiting any implied license or other defenses to infringement that may otherwise be available to you under applicable patent law.

12. No Surrender of Others' Freedom.

If conditions are imposed on you (whether by court order, agreement or otherwise) that contradict the conditions of this License, they do not excuse you from the conditions of this License. If you cannot convey a covered work so as to satisfy simultaneously your obligations under this License and any other pertinent obligations, then as a consequence you may not convey it at all. For example, if you agree to terms that obligate you to collect a royalty for further conveying from those to whom you convey the Program, the only way you could satisfy both those terms and this License would be to refrain entirely from conveying the Program.

13. Remote Network Interaction; Use with the GNU General Public License.

Notwithstanding any other provision of this License, if you modify the Program, your modified version must prominently offer all users interacting with it remotely through a computer network (if your version supports such interaction) an opportunity to receive the Corresponding Source of your version by providing access to the Corresponding Source from a network server at no charge, through some standard or customary means of facilitating copying of software. This Corresponding Source shall include the Corresponding Source for any work covered by version 3 of the GNU General Public License that is incorporated pursuant to the following paragraph.

Notwithstanding any other provision of this License, you have permission to link or combine any covered work with a work licensed under version 3 of the GNU General Public License into a single combined work, and to convey the resulting work. The terms of this License will continue to apply to the part which is the covered work, but the work with which it is combined will remain governed by version 3 of the GNU General Public License.

14. Revised Versions of this License.

The Free Software Foundation may publish revised and/or new versions of the GNU Affero General Public License from time to time. Such new versions will be similar in spirit to the present version, but may differ in detail to address new problems or concerns.

Each version is given a distinguishing version number. If the Program specifies that a certain numbered version of the GNU Affero General Public License "or any later version" applies to it, you have the option of following the terms and conditions either of that numbered version or of any later version published by the Free Software Foundation. If the Program does not specify a version number of the GNU Affero General Public License, you may choose any version ever published by the Free Software Foundation.

If the Program specifies that a proxy can decide which future versions of the GNU Affero General Public License can be used, that proxy's public statement of acceptance of a version permanently authorizes you to choose that version for the Program.

Later license versions may give you additional or different permissions. However, no additional obligations are imposed on any author or copyright holder as a result of your choosing to follow a later version.

15. Disclaimer of Warranty.

THERE IS NO WARRANTY FOR THE PROGRAM, TO THE EXTENT PERMITTED BY APPLICABLE LAW. EXCEPT WHEN OTHERWISE STATED IN WRITING THE COPYRIGHT HOLDERS AND/OR OTHER PARTIES PROVIDE THE PROGRAM "AS IS" WITHOUT WARRANTY OF ANY KIND, EITHER EXPRESSED OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE ENTIRE RISK AS TO THE QUALITY AND PERFORMANCE OF THE PROGRAM IS WITH YOU. SHOULD THE PROGRAM PROVE DEFECTIVE, YOU ASSUME THE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION.

16. Limitation of Liability.

IN NO EVENT UNLESS REQUIRED BY APPLICABLE LAW OR AGREED TO IN WRITING WILL ANY COPYRIGHT HOLDER, OR ANY OTHER PARTY WHO MODIFIES AND/OR CONVEYS THE PROGRAM AS PERMITTED ABOVE, BE LIABLE TO YOU FOR DAMAGES, INCLUDING ANY GENERAL, SPECIAL, INCIDENTAL OR CONSEQUENTIAL DAMAGES ARISING OUT OF THE USE OR INABILITY TO USE THE PROGRAM (INCLUDING BUT NOT LIMITED TO LOSS OF DATA OR DATA BEING RENDERED INACCURATE OR LOSSES SUSTAINED BY YOU OR THIRD PARTIES OR A FAILURE OF THE PROGRAM TO OPERATE WITH ANY OTHER PROGRAMS), EVEN IF SUCH HOLDER OR OTHER PARTY HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES.

17. Interpretation of Sections 15 and 16.

If the disclaimer of warranty and limitation of liability provided above cannot be given local legal effect according to their terms, reviewing courts shall apply local law that most closely approximates an absolute waiver of all civil liability in connection with the Program, unless a warranty or assumption of liability accompanies a copy of the Program in return for a fee.

END OF TERMS AND CONDITIONS