Renaming and Restructuring Files and Folders (v4.6)

The two most common questions we get are "Can Yate rename my files to..." and "Can Yate restructure my folders as...". The answer to both questions is yes.

Apple Apps and Renaming

Unless you configure iTunes, Music and the TV application correctly, it will mess up anything you want to achieve by renaming your files.

In the iTunes-Preferences-Advanced you must uncheck **Keep iTunes Media folder organized**. If you do not do so, iTunes will always name your files as it sees fit. Further, if the iTunes Advanced Preference **Copy files to iTunes Media folder when adding to library** option is set, iTunes will copy the files to its own notion of a folder structure.

In the Music application the equivalent settings are found in Music-Preferences-Files

In the TV application the equivalent settings are found in TV-Preferences-Files

If you are going to rename or move your files around, make sure that the tracks are linked to iTunes/Music/TV when you do the changes. This is necessary so that the application's library gets correctly updated. Note: if your tracks have retained Apple App PIDs (recommended) you can always refresh files when next linked to update a new location.

What's the Difference Between Renaming and Moving?

At the system level there is not much that is different ... everything is a Move. The differences appear due to the semantics applied when the functions are executed. One big difference is that volume changes cannot occur when renaming. Moves can be from any source to any destination. Moves create non existant path components. Renames change the name of existing path components.

If no folder names are modified, added or removed the two functions are essentially identical. When the number of folder components is changed or the actual location of a file is being changed, a Move must be used.

At times it is advantageous to move files as opposed to renaming them. Renaming always preserves the original folder structure and simply changes names. Regardless as to how many files are in a renamed folder, they always remain together. Moves, however, can put different files in different folders, creating the folders if required. Renaming only makes sense when all files in a folder have the same metadata used to name folders.

When files are moved, a single file is moved at a time. This is done in complete isolation to any other loaded file. After a file is moved it is entirely possible that the new and old folders exist. Even if you move all the audio files in a folder to another, non audio files will not automatically follow. (Yate does have an action statement to handle this situation named **Move Non Audio Files**). Other than automatically creating path components when necessary, you are really only working with a file.

When files are renamed you are working with individual path components. Yate renames (actually moves) each modified folder path component and file individually. All loaded files, whether being renamed or not must be adjusted for renamed folder path components. Here's an example. You have ten files loaded which are in **.../Music/Folder1/Folder2**. You select one file and rename its filename and at the same time rename Folder2 to be the Album field and Folder1 to be the Artist metadata. While the order of operations may vary this is essentially what happens:

- the filename of the selected file is renamed.
- Folder2 is renamed to the Album metadata. Every loaded file, whether selected or not, that resides in **.../Music/Folder1/Folder2** is marked as being relocated to **.../Music/Folder1/*Album***.
- If any of the files being processed is linked to an Apple application, the application needs to be notified.
- Folder1 is renamed to the Artist metadata. Every loaded file, whether selected or not, that resides in **.../Music/Folder1/*Album*** is marked as being relocated to **.../Music/*Artist*/*Album***.
- If any of the files being processed is linked to an Apple application, the application needs to be notified.

Because Yate is renaming the folder path components, the issue of files *not being moved* does not exist. However, if audio files in Folder2 have different Artist or Album metadata, Folder1 and Folder2 may be renamed multiple times. When done, the original files will always be in the same folder.

The choice of renaming or moving files is one of function. Basically remember that when moving a file only the file is being moved. When renaming a file, the file may be renamed as well as individual path components.

Yate v6.15 introduced a **Create Move Action Wizard** which can generate an action to move files based on a rename template. This action is automatically placed on the various **Rename>Actions** menus so that it can be accessed from the UI as easy as any rename template. These *move* actions are created by selecting a single rename template in **Preferences - Rename** and then hitting the blue action button.

Templates and the Token Editor

Yate uses rename templates to define what happens when files get renamed. These templates can be accessed via the UI or from within actions. The templates are defined in the Rename section of the application preferences. Many of the standard metadata fields can be directly applied in the templates and are sufficient for most renaming needs via the UI. When highly conditional or specific extracted data is required, it may be necessary to write an action to do the renaming. Rename templates can access Track Variables.

The rename templates are created in Yate's token editor. The use is quite simple. You add text and tokens wherever you see fit. You have absolutely no control over the filename extension and do not have to worry about it. In the token editor, tab characters and newline characters are ignored and can be used for formatting to make the template easier to read. Space characters are relevant and are displayed as centered dots so that they are easier to identify.

Post Rename Functions

Most people do the majority of their text manipulation prior to renaming. However, every rename template allows you to specify that certain functions should be applied after the new name is constructed.

If you want to perform alphabetic case manipulation, the process can be controlled by the **Case**, **Field Exception**, **Name exceptions**, **General exceptions**, **Replacement** and **Roman #** options. We're not going to look at the case transformations in this document. There is a Yate help topic called **Alphabetic Case Transformations** which describes the options in detail.

The **Spaces to _** setting will do as the name implies, changing all space characters in the constructed filename to underscore characters.

Macs support just about every possible character in filenames (almost). There are cases where you might want to have the characters fall entirely in the ISO Latin-1 character set. One possible reason is that you want to place your tracks into m3u files and your player does not support the m3u8 variants. The **Force Latin-1** option will ensure that only Latin-1 characters are in the constructed name. It does this by finding somewhat equivalent characters and by dropping characters if required. You can alternately **Force ASCII** which reduces the number of available characters to an even smaller set. If you wish you can not change the character encoding but simply remove accents from characters.

The **File Extension** settings are used to tell the Finder how to display the filename extensions. You can choose to leave the display option unchanged or you can force the display or hiding of extensions.

What Characters Fall into the *Almost* Category

Currently Yate always replaces **:** and **/** characters when renaming files. The text that replaces the characters is specified in the **Preferences - Rename - Settings - Invalid character substitution** setting. While you can create filenames in the Finder with embedded / characters, it is not a good idea. The APIs that Yate has to use do not differentiate between an embedded / and a / which is a path delimiter. Further certain APIs return /s as **:**s and it is not consistent across all APIs.

Yate also always removed leading periods from path components and filenames. A leading period is used to denote hidden on Macs.

Extraneous Spaces

You can also configure how Yate handles extraneous spaces when renaming. The **Preferences - Rename - Settings - Space handling** setting allows you to control how spaces are modified in each manipulated path component when renaming, copying or moving files. The choices are as follows:

No Changes
: Spaces are not modified.

Remove Leading/Trailing
: Leading and trailing spaces are removed.

Remove Leading/Trailing and Compress
: Leading and trailing spaces are removed and sequences of multiple spaces are replaced with a single space. This is Yate's initial setting.

Advanced Post Processing Settings

The rename token editor has a gear button at the top right of the panel. When you click the button, an advanced settings panel is displayed. Note that the circle icon to the left of the gear button is grey when no advanced settings are enabled and green when at least on advanced setting is enabled.

The advanced settings allow overrides and additional granularity for global rename settings.

While it is possible to have a replacement set which removes characters invalid in Windows, the advanced setting panel has a setting which enables the substitution of **\ * ? " < > |** characters with an underscore (_).

You have complete granularity of removing leading spaces and underscores; trailing spaces, underscores and periods; compressing sequences of spaces and underscores. When compressing, spaces and underscores can be handled separately or combined. The sequences can be replaced by a space, an underscore or the first character in the sequence being replaced.

The advanced settings optionally enables special processing when inserting text for duplicate filenames. This is discussed later.

Note that if any of the advanced rename settings are enabled, the default handling of spaces in filenames as defined in **Preferences - Rename - Settings - Space handling** is disabled.

Simple Renaming

Let's say you want to rename your files to start with the track number padded to two digits followed by a space and the tracks's title. Simply enter the appropriate tokens and text as follows:

> **(Track Pad2)·(Title)**

That's it! Notice the centered dot between the two tokens. That's the space character.

Conditional Insertion

Assume you want the template above to be modified to insert the disc number but only if it exists. In this case *exists* means not empty or zero. If it does exist we want the disc number followed by a dash.

> **(IfExists Disc)(Disc)-(endIf)(Track Pad2)·(Title)**

The **IfExists *field*** token implies that the following text and tokens should only be applied up to the next **Else** or **endIf** token, if the field exists. If the field does not exist and an **else** token is supplied, all text and tokens after the **Else** until the matching **endIf** will be processed. In the above example if Disc exists, we're inserting the **Disc** token and a dash character.

Tokens which begin with the following names treat a numeric value of 0 as being equivalent to empty: Disc, Episode, Movement Number, Season and Track. Any custom field defined as being a Yes/No value will also treat a numeric value of 0 as equivalent to being empty.

As a convenience, there is a special form of the IfExists token which says *if the specified field exists, insert it immediately after the token*. The previous template can be simplified to:

> **(IfExists+ Disc)-(endIf)(Track Pad2)·(Title)**

As a somewhat silly example, let's say we want to specify **ND** if there is no disc number.

> **(IfExists+ Disc)(Else)ND(endIf)-(Track Pad2)·(Title)**

Using the Format button in the token editor, we can make the text somewhat easier to read.

```
(IfExists+ Disc)
(Else)
    ND
(endIf)
-(Track Pad2)·(Title)
```

Here's a more complicated example which inserts the disc and track number if the disc field exists. If disc does not exist and track does, the track number is inserted. If either sequence was inserted, a space will be added before the Title.

```
(IfExists+ Disc)
    -(Track Pad2)·
(Else)
    (IfExists+ Track Pad2)
        .
    (endIf)
(endIf)
(Title)
```

Restricting the Length of a Name

If you wanted to limit the length of a name to 100 characters, you could do the following:

> **(Title)(IfLength)100(Truncate)99(Break)…(endIf)**

The above template inserts the Title field and then tests if the name is longer than 100 characters. If it is, the name is truncated to 99 characters and a horizontal ellipsis (...) is inserted. The (Break) token does not insert anything. It serves as a token which can be used as terminator of a text field or to separate two text fields. In the above example the **IfLength** and **Truncate** tokens read the following text field for the *number*. If for any reason, *number* is followed by text a (Break) token must be used.

There is also a **Preferences - Rename - Settings - Automatically truncate path components** setting. It can be difficult to determine if a filename or path component is too long for an underlying filesystem. The setting attempts to remove some of the burden. When set, filenames and path components will be reduced to fit within 255 bytes based on the system's representation. Note that due to encodings, a truncated component may be less than 255 characters. Bytes were chosen as a unit, as APFS is based on UTF8 characters, HFS+ is based on UTF16 characters, etc. The setting errs on the side of caution. It is possible that extra characters can be manually entered when editing a path component in the Finder.

Duplicate Handling

It is entirely possible that when you rename files you will end up with a duplicate filename. For example, if you're renaming files to the Title field, and more than one track is named **Intro**. Yate allows you to handle duplicate filenames in the template. You can insert whatever text and/or tokens you wish inside of the duplicate sequence. However, it is typically better to let Yate pick a single number to differentiate the copies.

> **(Title)**
> **(IfDup)**
> ·**(Dup #)**
> **(endIf)**

The above template says that the name consists entirely of the Title field. If it is a duplicate, add a space and the first available *duplicate* number.

Duplicate processing, can cause issues with removing trailing spaces and underscores as a duplicate sequence can be added at the end of the path component or filename. The advanced preference setting **Apply transformations for IfDup**, applies various transformations before the insertion point of a duplicate sequence. more information can be found on the help for the advanced rename settings panel.

Multiple Values in Fields

There are two methods of handling multiple values in fields. If the **Preferences - Rename Multi Value Delimiter substitution** setting is empty, all values following a multi value delimiter (typically ;;;) are ignored. If the setting is not empty, multi value delimiters are replaced with the specified text. As an example, if you specify **" and "** as the rename substitution, all **;;;** sequences will be replaced with **" and "**.

You can also use a **(Multi Value Delimiter)** token to override the *rename substitution* setting in a template. These tokens can be specified as many times as desired in a template to provide complete granularity on how multi value delimiters are processed. If the token is followed by text, the text becomes the replace substitution value. If the token is followed by another token, the replace substitution value is set to empty. The following example overrides the *default* value only for the Album Artist field.

> **(Title) - (Multi Value Delimiter) and (Album Artist)(Multi Value Delimiter)(Break) - (Genre)**

Note that the **(Break)** token is used to force a *no text* situation before the " - " text which is being inserted before the genre.

You can also test if a field contains multiple values. One possible use of this is that you possibly only want to include a field if it does. The **IfMultiple *field*** token is another if-then-else token which tests if the specified field has multiple values. The following example only includes the Artist field in a filename if it contains multiple values. A **,** is used to separate the multiple values.

```
(IfMultiple Artist)
   (Multi Value Delimiter),(Artist)·-·(Multi Value Delimiter)
(endIf)
(Title)
```

Renaming Folders

In much the same way that you rename files, you can rename folders. You rename files and/or folders in a single template. The *folder renaming* portion of a template follows the filename portion of the template. The **Folder Start** token stops all filename processing and begins folder renaming. Any open ***if*** tokens are automatically closed. As an example, the following template renames the containing folder to be the Album name and the filename to be the Title.

```
(Title)(Folder Start)(Album)>
```

Note that duplicate handling tokens can only be placed in the filename portion of the template. A more complete example which does the previous *disc-track* based filenames, adds duplicate handling and renames the folder to *Year-Album* follows:

```
(IfExists+ Disc)
    -(Track Pad2)·
(Else)
    (IfExists+ Track Pad2)
        .
    (endIf)
(endIf)
(Title)
(IfDup)
    ·(Dup #)
(endIf)

(Folder Start)
(IfExists+ Year)
  .
(endIf)
(Album)
```

You can rename more than one path component by specifying multiple **Folder Start** tokens. All renaming is done right to left in a path. The start of a template defines the structure of a filename. The text and tokens after the first **Folder Start** token specifies the name of the folder containing the file. The second **Folder Start** token renames the folder containing the folder which contains the files. etc.

You must define something after a **Folder Start** token otherwise at runtime you will get an empty path component error. If you do not wish to modify a path component which is followed by subsequent **Folder Start** tokens, specify an **Initial Folder** or **Current Folder** token. Here's a simple example which renames files to the commonly used ***.../Artist/AlbumTitle*** format:

```
(Title)
(IfDup)
  (Dup #)

(Folder Start)
(Album)

(Folder Start)
(Album Artist)
```

If there are more **Folder Start** sequences specified than there are available path components, they are silently ignored at runtime.

If you know that a path component is optional and can end up being empty, you can specify an **Ignore if Empty** token in the **Folder Start** section. When **Ignore if Empty** is present and a path component ends up being empty, the **Folder Start** section is ignored.

Sorting Considerations

All of the sort fields contained in Yate are available as tokens. If you are not maintaining the sort fields there may still be times when you want to remove leading articles from a field. The **Remove Leading Articles** token causes a single leading article, if present, to be removed from all subsequent inserted fields. The list of articles which are to be removed is defined in the Preferences - Exception Natural Sort Set. To stop removing articles, you can specify a **Preserve Leading Articles** token. For example the following template renames the containing folder to artist - title removing a leading article from the artist metadata:

(Folder Start)(Remove Leading Articles)(Artist)(Preserve Leading Articles)·-·(Album)

Special Field Handling

The **Album Artist** token will insert the **Artist** field if the **Album Artist** field is empty.

The **Year** token respects the **Format the Year field** setting as defined in Preferences - File List. The **Year4** token always inserts the year as YYYY.

There are no Tokens for What I Need!

It is entirely possible that the filename you want to construct cannot be specified with a simple rename template. Yate handles this case by allowing you to place **Variable 0** through **Variable 15** tokens in a template. This allows you to write an action which constructs the *special purpose* portions of the desired filenames and places them in variables. These variables are then accessed by the template. Let's assume you want to rename the folder containing an album to the album name optionally followed by some text specified at runtime enclosed in square brackets. You could use the following template and action:

```
(Folder Start)

(Album)
(IfExists Variable 1)
   ·[(Variable 1)]

Prompt for Text save to Variable 1, "Specify text to be used when renaming"
Trim Variable 1 (SP) [Leading] [Trailing]
Rename Files (above template) silent
```

When writing an action to rename files you might have a variable which is essentially true or false to be used as a test condition. Similar to the IfExists token, you can use IfTrue and IfFalse tokens with track variables. As an example, assume that the Title is to be followed by the Artist but only if it is not the same as the Album Artist field:

```
(Title)
(IfTrue Variable 1)
      ·-·(Artist)
(endIf)

Test if the Album Artist field is not equal to "\[Artist]" (Set result and Variable 1)
Rename Files (above template) silent
```

Here's a tip. When you have actions such as the ones above which essentially simply do renaming, you can place the action on the rename menu. In the Action Manager, select the action and from the context menu choose, **Add to Menu>Rename** (or click in the Menu column).

Moving your Files to a New Folder Structure

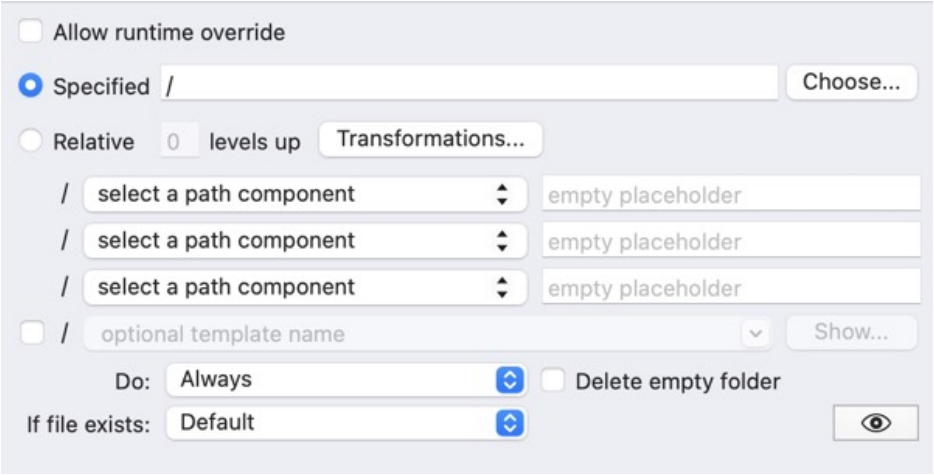
This section discusses how to perform *moves* from within action. As previously mentioned, Yate v6.15 introduced a **Create Move Action Wizard** which can generate an action to move files based on a rename template. This action is automatically placed on the various **Rename>Actions** menus so that it can be accessed from the UI as easy as any rename template. These *move* actions are created by selecting a single rename template in **Preferences - Rename** and then hitting the blue action button.

There appears to be an infinite number of folder structures that people use for their collections. Whatever works for you is great. However, here's an opinion. Many people like to separate multi-CD albums into different folders based on the disc number. This is an unnecessary complication. There is a Disc field which when used with the Track field will uniquely identify where the track came from. There are some high end audio players which do not like having albums split into separate folders. Nor do they necessarily handle Album names differentiated by appending Disc 1, Disc 2, etc. We recommend having a single consistant Album field for all tracks in an album. Use the Disc fields as intended. Generally, we feel that life is easier with the one album / one folder approach. However, if you want separate disc folders, go right ahead...Yate will produce them for you.

The actual movement of tracks is done in actions via the **Move** statement. If you are re-structuring, you will more than likely want to move any non-audio files to the same new folder structure. This is done by the **Move Non Audio Files** statement.

The Copy Files and Move Statements

Here's what the Move statement looks like:



There are lots of options and fields, but the statement does a lot.

The first decision you have to make is whether you want the tracks to be moved to a location relative to where they currently reside or do you want to place them *under* a fixed location. While the relative method has its uses it is far more common to have your collection *rooted* in a single well known location. For example: **/Users/me/Music** or **/Volumes/MyMusicCollection**. You get the idea. You can directly specify the absolute path or you can construct a path at runtime by using escape sequences. This allows you to issue a prompt for the path if you so desire. As an example, a specified path of **\v1** will take the path from each track's **Variable 1** field.

You can now choose to append up to three additional path components which are extracted from a track's fields. If a component is empty at runtime an error will be issued. You can get around this problem by specifying a *placeholder* to be used if the specified field is empty. If you specify the **Album** field and a placeholder of **Unknown Album**, the path component will be **Unknown Album** if the **Album** field is empty. If you wish you can skip the field entirely and simply specify a placeholder. As a convenience, if the **Album Artist** field is specified and it is empty at runtime, an attempt will be made to use a non empty **Artist** field before falling back to the placeholder.

Regardless as to how you specify the path and additional path components, the resultant path at all levels will be created if it does not already exist.

What about the filename? If you don't specify a rename template, the original filename will be used. However, if you do specify a rename template, the template will formulate the final appearance of the filename. Further, if the template contains one or more **Folder Start** tokens, the folders constructed in the template will be appended as path components. This means that you can effectively construct **three +** ***the number of Folder Start*** path components based on metadata. .... and as an added bonus, if the rename template contains at least one **IfDup** sequence, the specified duplicate file semantics will be applied if necessary.

As it is quite common to structure your compilations differently, there is a option to execute the **Move** statement only if the track has **Part of a Compilation** field set or not set. This means that you can have two back to back Move statements where only one of the two executes based on the **Part of a Compilation** setting.

The most common file structure by far is someting like: ***/collection*/Artist/Album/*files***, where Artist and Album may contain additional metadata other than what the name implies. Many people handle their compilations *out of bounds* as ***/collection*/Various Artists/Album/*files*** or ***/collection*/Compilations/Genre/Album/*files*** or something else entirely. The advantage is that compilation albums can be kept in a single folder as opposed to being placed by artist and scattered all over the place (which is much what iTunes and Music does). Being able to trigger on the **Part of a Compilation** field allows you to transparently handle both potential placements.

Most people handle the metadata text manipulation external to the Move statement. However, if you're the type of person who wants all spaces replaced by underscore characters, you can do so in the Move statement. This is largely a throwback to earlier days. Macs have no issues with spaces in filenames.

The Move statement allows you to specify what to do if a file already exists at the target location. Initially the Move statement did not support duplicate handling via a specified rename template. We feel that the rename template method of handling duplicates is far superior. It'll work while batch processing, doesn't pop up dialogs and ensures that a unique filename can be constructed.

You can tell Yate to delete the old folder and possibly the parent folder, if they are empty after the move. This tends to clean up a lot of empty folders which can clutter up your drive and be annoying and misleading. More on this option later.

The Copy Files statement is identical to the Move statement except that ... files are copied, not moved.

Both the **Copy** and **Move** statements have a **Transformations** button which displays a panel of additional settings which replication most of the rename transformations ... only these are used for the path components.

At times, you may have to decide at runtime if you want to move or copy files. The **Move** statement has an **Allow runtime override** setting. If the setting is enabled and named variable **Override Move to Copy** is true (non zero integer or "true"), the **Move** statement will effectively become a **Copy Files** statement. When overridden the folder deletion settings will be ignored.

The Copy Non Audio Files and Move Non Audio Files Statements

When relocating tracks, it's may be desirable to move non audio files such as images, PDFs, etc. along with the audio files. In Yate you do this with the **Move Non Audio Files** statement. Here's what the statement looks like:

You can attempt to move all non audio files or explicitly specify which files or types you want to move or exclude from moving. If you want to move everything, leave the two text fields empty. The inclusion and exclusion fields all take a list of items separated by / characters. An item can be a filename or file extension. The statement is smart enough to not require periods. **jpg** will match all **.jpg** files. While the inclusion/exclusion capability is pretty comprehensive, most people tend to move everything.

Items in the list can be forced to match only file types by starting the item with a period. Items in the list can be forced to match only files by ending the item with a period. Forcing the match type should only be necessary in extreme cases. For example:

txt./.jpg

The above example will match files named txt and any file with a file type of jpg.

In the inclusion field you can also specify a rename pattern for files being processed. This is only valid for fully qualified filenames or folder names. You specify the new name following a : character. For example to rename files named cover.jpg to folder.jpg you would specify the following:

cover.jpg:folder.jpg

Unless you explicitly say that you want subfolders moved, they will be ignored. From the documentation:

> A subfolder can be moved if none of its descendents is an audio file. When you are attempting to move subfolders, all subfolders will be processed (unless hidden). When processing a subfolder the inclusion list is ignored, however the exclusion list is still processed. This means that when attempting to move subfolders, all subfolders in a source folder will be processed unless they are explicitly named in the exclusion list. Note that contents of a sub folder are always moved if the subfolder is moved. The inclusion and exclusion lists are ignored for subfolder contents.

If you read the above about four or five times, it will probably make sense. If you're moving subfolders, they're all moved unless explicitly indicated that they shouldn't be moved. The contents of a moved subfolder is always moved.

Yate has to know what folder you're moving from and what folder you're moving to. In order to accomodate different scenarios, there are three different ways to specify the folders:

Folders from variables
: This one is pretty easy to understand. As every track has a unique set of variables, the folders may be different for each track.

Use retained paths (set by Move)
: The Move statement, on a file by file basis, saves the *before* and *after* paths to a track in the **Retained Path 2** and **Retained Path 1** properties. Now be a little careful here as a few other statements, such as **Rename**, set the same properties. If a **Move Non Audio** statement comes after a **Move**, the non audio files will move from the *before* to *after* folders.

Use original and current folders
: A special case scenario when the *original* folder is the location of an audio track when first loaded (even if it has been subsequently moved a few times). As long as a track stays *loaded*, the *original* folder does not change. The *current* folder is the track's containing folder.

Regardless as to which folder specification method you choose, Yate will ignore the last path component if it is an audio file. That's how the retained paths method works, as the properties always contain the full path to the file.

The **Move Non Audio Files** statement is pretty efficient. If will only handle a given source folder once. Further, if a source folder somehow ends up being associated with more than one destination folder, nothing wil be moved. This can happen if tracks in the same folder end up dispersed to different folders.

As with the **Move** statement, you can specify that a source folder and its parent folder should be deleted if they empty out. If you're going to follow a **Move**, by a **Move Non Audio Files** statement, there is no benefit to doing it twice...but do it on the **Move Non Audio Files** statement as it more than likely comes last.

By default, existing files will be overwritten. If you want to preserve existing files in the case of duplicate names, there is a Preserve existing files setting.

The Copy Non Audio Files statement is identical to the Move Non Audio Files statement except that ... files are copied, not moved.

At times you might want to decide at runtime if you want to move or copy the files. The **Move Non Audio Files** statement has an **Allow runtime override** setting. When the setting is enabled, named variable **Override Move to Copy** is examined. If the variable is true or a non zero value, the **Move Non Audio Files** statement will be treated as a **Copy Non Audio Files** statement. When overridden the folder deletion settings will be ignored. Override Move to Copy

Tips and Tricks

At times you may have to execute the **Move** statement *grouped* as a number of conditional choices may have to be made regarding the destination. There is not really much of a difference, when executing a Move *stepwise* or *grouped*. Checking to if the source folders can be deleted is a slight overhead, but only a slight one.

The **Move Non Audio Files** statement can be far more efficient when executed *stepwise*. There is a lot of logic in place to handle overlapping moves and directory scans.

If you've moved a track to a separate Disc subfolder, under the album folder, you more than likely want the non audio files in the album folder as opposed to the disc folder. A neat little trick is to simply remove the last two path components from the **Retained Path 2** property set by the **Move** statement. The last path component will be the filename and the second to last will be the Disc folder. Here's a little snippet which will work:

```
Get Property Path to Folder -> Variable 1
Set Variable 0 to "Disc \[Disc]"
Move to /MyMusicCollection/Artist/Album/Variable 0/{filename}
Get Property Retained Path 1 -> Variable 2
Remove the last path component in Variable 2
Remove the last path component in Variable 2
Move all non audio files from Variable 1 to Variable 2
```

The above is a somewhat simplistic approach as it always creates Disc subfolders and doesn't validate the disc number, but it will work as is.

Summary

The ability to rename and to relocate files is pretty complete. We haven't found any scenario so complicated that an action could not perform the desired operation. Play with Rename templates and the Move action statement. Both functions have a preview capability.

Additional information, including sample actions can be found at Yate Resources. You can also visit the forum.

History

Date	Version	Information
2016-09-17	1.0	First release.
2017-12-10	1.1	Documented the Remove accents template option.
2018-04-03	2.0	Updated for changes in Yate v4
2018-04-08	2.1	Documented the enhanced handling of fields with multiple values.
2018-06-09	2.2	Minor tweak to Move Non Audio example to use newer statements.
2019-09-10	3.0	Changes for Yate v5.0
2020-07-09	4.0	Changes for Yate v6.0
2022-06-07	4.1	Changes for Yate v6.10.2
22-07-06	4.2	Changes for Yate v6.10.5
2023-04-11	4.3	Changes for Yate v6.15
2023-04-30	4.4	Changes for Yate v6.15.1
2023-06-30	4.5	Changes for Yate v6.16
2023-08-14	4.6	Changes for Yate v6.17.1