Track Parser 1.3.1

By Dan Vanderkam
June 29, 2009

What's it Good For?
Track Parser is a collection of AppleScripts which are designed to parse song names for information, greatly simplifying the process of tagging every song you import. In most cases, you've already got all the information that you'll be entering in some form or another, be it an info file that came with your download, or in the file names. Adding that information yourself is sheer drudgery. Track Parser lets you specify a format, and then hand over all the work to the computer.

Say you've just imported some songs, and their names appear in "artist - album - track - name" format, as follows:

```
My Bloody Valentine - Loveless - 1 - Only Shallow
My Bloody Valentine - Loveless - 2 - Loomer
.
.
.
My Bloody Valentine - Loveless - 12 - Soon
```

You'd like to parse these track names into the four separate fields. Usually, this would be a real pain. Before writing this script, I would have:

1. Used the "Get Info" dialog to set the artist and album
2. Run "Remove N Characters from Front" to get the track numbers in front
3. Run "Put Track Prefix to Track Number"
4. Run "Remove N Characters from Front" again to get rid of the remaining " - "

That isn't too bad, but it's needlessly involved. To do the same thing with Track Parser, you'd simply run "Parser (Song Names)" and pick the appropriate pattern from the examples, in this case "`The Beatles - Revolver - 01 - Taxman`", since this matches the format above.

That's it. The script will do the rest of the work for you, setting all four fields. Most of the formats you'll run into are represented in the set of formats that comes with Track Parser. If your's isn't though, you can create your own. Just scroll down to the **Creating Your Own Patterns** section.

Other Uses
Track Parser can do its work in a variety of settings. In particular, it can take its data from the clipboard rather than from the song names. I've found this most useful for live music that I've downloaded. The tracks are typically named something along the lines of "pixies2004-04-19d1t01", and there's an accompanying text file which contains the actual song names:

```
01. Bone Machine
02. River Euphrates
...
11. Planet of Sound
12. Into the White
```

To handle this common situation, I'd select all the songs, run "Parser (Song Names)", and choose the appropriate pattern, in this case "`pixies2004-04-19[-]d1t01`". Yes, this might just be where that format got its name...

This will fill in the artist (pixies), album (2004-04-19), year (2004), disc (1), and track (01) fields. But not the song name, which happens to be pretty important. So copy the set list to the clipboard, and sort the tracks in iTunes. In this case, you can just sort by song name. All that matters is that they appear in the same order in iTunes as they do on the clipboard. Then run "Parser (Clipboard)" and select the appropriate format, in this case "`01. Airbag`". If you've done everything correctly, then all the song names should be set.

Substitutions
Track Parser can also clean up text within a single field. Specifically, you can do find/replace on a song name using regular expressions. See below for details on this.

The one example that I've included is a Title Case converter. It takes nasty looking track names like "THE WIND IN THE WILLOWS" and changes them to "The Wind in the Willows". It does a fairly good job, though it's certainly not perfect. It's likely to mess up acronyms like "DJ", which will get converted to "Dj".

This mode has some additional features which make it extremely powerful. See below for examples of its use.

The Manager
If you'd like to change, modify, or remove any of the patterns that appear in the favorites list, then run the "…(Manager)" script. You'll be asked which favorites list you'd like to modify, and then presented with a menu of options. This also provides a means to see the patterns that the existing favorites are using.

If there's a pattern that you find yourself using frequently, you can use the manager to spin it off into its own separate script. This way, it will appear directly under the Scripts menu in iTunes, saving you a step. Once it's in the Scripts menu, you can assign it a keyboard shortcut to be even more efficient.

Creating Your Own Patterns

I've tried to include most common formats in the built-in list, but if I missed yours, then this section is for you.

Simple (scanf-style) Patterns
For a simple pattern, you can use a format string reminiscent of C's *scanf*/*printf* functions. All text is matched literally, unless it's a character

proceeded by a %-sign. These metacharacters denote the individual fields. So, for example, "`%t - %n`" would match "01 - Airbag". The " - " is matched literally, while the `%t` and `%n` suck in everything else, and reroute whatever they take to the appropriate field. Here's the complete list of characters than can follow a %-sign.

Code	Field
`a`	Artist
`A`	Artist (no clobber)
`b`	BPM
`c`	Comments
`C`	Composer
`d`	Disc Number
`D`	Disc Count
`e`	Played Date
`f`	File Name (read-only)
`g`	Genre
`G`	Genre (no clobber)
`l`	Album
`L`	Album (no clobber)
`n`	Song Name
`p`	Played Count
`r`	Grouping
`R`	Grouping (no clobber)
`t`	Track Number
`T`	Track Count
`y`	Year
`0`	Nothing (skip the text)
`#`	Rating (Between 0 and 100)

As I mentioned earlier, anything character not prefaced by a "%" sign is matched literally. This includes spaces and punctuation, since they're usually what delimits fields in song names. The "no clobber" entries will only change a field if it's blank -- they won't clobber any existing information.

Here are some common formats and their corresponding patterns, to help you get the hang of this:

Pattern	What it Matches
`(%t) %n`	(01) Airbag
`%a - %t - %n`	U2 - 06 - Red Hill Mining Town
`%a - %l - %t - %n`	XTC - English Settlement - 15 - Snowman
`%a - %l - CD%d - %t - %n (%y)`	Bob Dylan - Live 1966 - CD1 - 01 - She Belongs to Me (1966)
`%a2%0d%dt%t`	pixies2004-04-19d1t01 (doesn't get the year part, see below)

Note that text and tags are always appear one after the other. The parser won't know what to make of something like `%t%n`. This situation doesn't come up very often, but as the last example shows, it certainly could. In that case, you'll need something a little more powerful. Namely, …

Regular Expressions
Whenever the parser sees a pattern prefaced with '/' (a forward slash), it assumes that it's a regular expression, and hands it off to perl (which, therefore, you need installed for this feature). This gives you an incredible amount of power to match complicated patterns. It's essentially inconceivable that your song names would be so convoluted that a regular expression couldn't describe them. But regular expressions are usually overkill, and they're more error-prone than the simpler patterns which I've already discussed.

The basic syntax of a regular expression in these scripts is:

    /pattern/fields

The parser applies the pattern to each name/line, keeps track of all the captured subexpressions, and then uses the fields list to find out what each subexpression is. Maybe some examples will help. I'll assume you're familiar with regular expressions. There are countless websites and books devoted to them if you're not, and they're a lot of fun to use.

Let's use the example from the beginning of the Read Me. The song is named:

    My Bloody Valentine - Loveless - 1 - Only Shallow

To match it, you might enter the following:

```
/^(.*?) - (.*?) - (\d+) - (.*)$/altn
```

This isn't the only pattern that you could use, but it certainly works. Any regex features that perl supports will work, like the minimal matching modifier '*?'. Each relevant piece of information is individually captured with parentheses. The "altn" afterwards means that the captured subexpressions are, in the order that they were captured: 'a' (artist), 'l' (album), 't' (track number), and 'n' (song name).

In Perl, expression modifiers like 'i' (case insensitive) usually appear after the final slash. So those are not supported by the parser. Patterns are, by default, run without case sensitivity. There are ways that you can toggle these modifiers from within a pattern in Perl. If you need to do that, check out *Programming Perl*, by Larry Wall, or the various Perl websites that are online.

But that's unlikely to ever come up. In most cases, the simpler *scanf*-style patterns will more than suffice, and you're less likely to make

mistakes writing them.

Here are some more examples of regular expressions used to match songs:

Pattern	What it Matches
`/\((\d+)\) (.*)/tn`	(01) Airbag
`/^(.*?) - (\d+) - (.*)/atn`	U2 - 06 - Red Hill Mining Town
`/^(.*?) - (.*?) - (\d+) - (.*)/altn`	XTC - English Settlement - 15 - Snowman
`/^(.*?) - (.*?) - CD(\d) - (\d+) - (.*)/aldtn`	Bob Dylan - Live 1966 - CD1 - 01 - She Belongs to Me (1966)
`/(\d\d)(.*)/tn`	01Subterranean Homesick Blues
`/(\D+)((\d+)-\d+-\d+)d(\d)t(\d+)/alydt`	pixies2004-04-19d1t01 (gets the year part!)

The last two examples are the most likely uses of regular expressions. The second to last example couldn't be matched with a *scanf*-style pattern, since there's nothing to seperate its two fields. The last pattern is fairly involved. The '2004' portion of the song name is actually captured twice: once for the year, and again as part of the album ('2004-04-19').

Substitutions
If Track Parser supports Perl's `m/../` syntax, then why not support its `s/../../` syntax? If your pattern starts with `s/`, then it will be passed to Perl essentially verbatim, as a substitution operator. Whatever results from the substitution will become the new track name. You can do find and replace this way, for example:

```
s/Radio Head/Radiohead/gi
```

The `/g` is necessary if you'd like to do a "Replace All", as opposed to just a "Replace". The `/i` makes it case-insensitive. Again, check out a Perl book for details. This feature is used mainly by the Title Case pattern at the moment, which is a real monster of an expression. Check it out in the manager.

With the /e modifier, you can execute Perl code in the replacement section, which gives you a lot of power, some of it useful, some of it not. For instance:

```
s/(.*)/$1/ee
```

evaluates each song title as a Perl expression, and replaces the title with the result. This is completely useless (except maybe for Radiohead's "2+2=5"), but it does illustrate the power of this feature.

Fancy Substitutions
Unlike the other pattern styles, substitutions can take their input from any tag (not just the song title or a line on the clipboard), and they can get user input before they run.

The full syntax for a substitution is:

`s[`*Question 1*; *Question 2*; *…*`]{`*altn…*`}/`*pattern*`/`*replacement*`/[msixeg]`

The brackets delimit a series of semicolon-separated questions. When the script is run, each of these will be asked in sequence before any matching is attempted. The results will be stored in `@ans`, an array accessible in both parts of the pattern. If no questions are supplied, no questions will be asked. This is a purely optional part of the pattern.

The tag identifiers inside the squiggly braces specify the tag(s) on which the pattern should be run. If none is supplied, a default of `n` (song name) is used. If multiple tags are specified, they are both fed into the pattern, separated by a newline (matchable as `\r` in the pattern). For instance, if the song was "Airbag" by Radiohead, off of *OK Computer*, and the pattern was:

```
s{al}/…/…/
```

Then the pattern would be matched against:

```
"Radiohead
OK Computer"
```

(the quotes aren't part of the string)

By default, when the substitution is done, Track Parser will put the resulting value back into the first source tag (`a` for artist in the above example). To override this default, return a value of the form:

```
"altn…
Value 1
Value 2
…"
```

The first line provides a key for all the others. The first letter identifies where the data in the next line goes. The second identifies where the line after that will go, etc. So for instance, the following pattern will swap a song's artist and album:

```
s{al}/^/la\n/
```

This is a cute trick. The pattern `/^/` matches at the beginning of the string, and inserts `la`, followed by a newline. So the output from the above Radiohead song is:

```
"la
Radiohead
OK Computer"
```

This says that the album (l) is Radiohead, and that the artist (a) is OK Computer. This is just what we wanted. You could use the same trick to achieve any permutation of tags. For instance, this pattern does a cyclic permutation of the song's name, artist, album, genre, grouping, and composer:

s{nalgrC}/^/Cnalgr\n/

Run that six times, and you should be exactly where you started.

As an example of how to use the questions, let's write a script that sets a song's artist and album. This might be useful if you insert a CD when you're not connected to the internet, or the CDDB database doesn't know about it. The script should ask for the artist, and then the album. Here it is:

```
s[Enter Artist:;Enter Album:]/.*/al\n$ans[0]\n$ans[1]/
```

The two questions are in square brackets, separated by a semicolon. The output consists of three lines. The first, `al`, says what the next two lines are. The next line is `$ans[0]`, the answer to the first question, and the last is `$ans[1]`, the answer to the second question.

Why is the pattern in this example `.*`? Because, by default, the text supplied to the substitution is the song name. That's totally useless to us, so we want to write over the entire string. Matching `.*` takes the entire string and replaces it with whatever we supply. This is stretching the idea of a substitution, but that's the syntax, and I'm not ready to add an entirely new syntax just yet.

For more examples of fancy substitutions, see the "Imitating Other Scripts" section, below.

Test Runs
With longer patterns, odds are you'll mess up at some point. That's what the "Test Run" button is for. If you click this, then instead of actually changing the track info, you'll just be informed of what would happen. This is especially important if your pattern changes the track's name, since you could potentially lose information if your pattern runs amok.

The Test Run button opens up a new dialog for each field that matches. Since this could potentially mean several dialogs per song, I'd suggest just running it on one of the songs, to save yourself the pain of clicking "OK" 50 times. Or just click "Cancel" in one of the dialogs if you're convinced that the pattern is working.

Spinning off Scripts
Using the "… (Manager)" script, you can take a commonly-used pattern and convert it into a standalone script. This gives you quick access to it in the Scripts menu, and it also allows you to assign it a keyboard shortcut using the Keyboard control panel. (there's an article on http://www.malcolmadams.com/itunes/ explaining this) If you download a lot of live music, then the live music pattern (the "pixies2004…" pattern from above) would be a likely candidate for spinning off.

Another way to think about this feature is as an alternative method to write scripts. Instead of writing your script using AppleScript, write it using Perl, regular expressions, and Track Parser. The patterns are flexible enough to encompass many existing scripts (see below), and if you know Perl, you'll save yourself a lot of time. The section "Imitating Other Scripts" details how about 40 of the scripts on Douglas Adams' site could have been written as one-liners using Track Parser.

Tips and Troubleshooting
These scripts (and iTunes scripts in general) seem to run faster when you're viewing a playlist, as opposed to your library. I've got a "Recently Added" smart playlist which displays the 200 or so most recently added tracks. I do most of my editing from that screen. Things also tend to go faster if you're not changing information that would cause iTunes to resort the songs. So if your playlist is sorted by track name, then a matching a pattern with `%n` in it might run slow. My "Recently Added" playlist is sorted by Date Added, which is read-only, so a script could never force it to resort.

Track Parser will run almost instantaneously on some tracks, but very slowly on others. As far as I can tell, the slowdown happens when iTunes has to open up the MP3 file and rewrite it to make the modifications. After this has been done once, any additional changes to the track will occur quickly.

Another use of the "…(Clipboard)" script that I've run into is when the file names of your MP3s contain all the right information, but the tags are wrong. In this case, you can select all the affected files in the Finder, and choose "Copy" from the "Edit" menu. This will copy a list of the files to the clipboard. You can then paste this into TextEdit to rearrange the lines in the correct order, before running the script.

Also, if your songs have helpful names like "TRACK 01", but you know the album, you could copy the track names from Amazon, clean them up a bit, and run the "…(Clipboard)" script on them.

The `%0` or `/0` field may look puzzling—why would you throw away information?—but it can actually be fairly useful. If you've already got information, like the album, track numbers, … and it's also contained in the track name, then there's no reason to do more work than necessary. A situation where this often comes up is when the track names are in the "`pixies2004-04-19d1t01`" format, and you have a set list with lines in the "`01. Bone Machine`" format. You've got the track number twice, so there's no need to use it twice. Because of the way that iTunes handles changes in ID3 tags, this is unlikely to result in a speed bump, but if the alternative bothers your aesthetic sense, at least the option's available…

Imitating Other Scripts
Track Parser is extremely flexible, and as such, it can perform the exact same functionality as plenty of other existing scripts. For example, looking through the scripts on Doug Adam's site, I noticed the following scripts. Take these as "real world examples":

Script	Track Parser Pattern
Remove n Characters from Front	`s[Remove how many characters?]/^.{$ans[0]}//`
Artist - Name Corrector	`%a - %n`
Remove Extension from Song Name	`s/\.(mp3|m4a|aiff?|wav|ogg|aac)$//`
'First Item - ' to Tag	`%[alt…] - %n`
Proper English Title Capitalization	(It's ugly, but included in the default list)
Track Name to Sentence Caps	(similar to the above pattern)
Put Track Prefix to Track Number	`/(\d{2})(.*)/tn`
Track Names to Word Caps	`s/\b(.*?)\b/\u\L$1/g`
Track Names to Sentence Caps	`s/(^.)|(.)/"\u$1" || "\L$2"/ge`
Zero-Pad Initial Digits	`s/^(\d+)/sprintf "%02d", $1/e`
Append to Selected Tag	`s[Text to append:]{`*tag*`}/$/$ans[0]/`
Number Prefix to Track Number	`/^((\d+).*)/tn`
Replace Text	`s[Text to find:;Replace with:]/\Q$ans[0]\E/$ans[1]/g`
Make 'Track' My String	`s[String to replace 'Track' with:]/Track/$ans[0]/`
Add Text at end (of Track Name)	`s[Text to add:]/$/$ans[0]/`
Strip Stuff From Tracks	`s/`*thing one*`|`*thing two*`|…//g`
Remember These Tags For A Sec'	(theoretically possible—it would use a substitution that read from/wrote to a file)
Append to Selected Tag	`s[Text to append:]{`*tag*`}/(.*)/$1$ans[0]/`
Append to Comments	(see above)
This Tag, That Tag Scripts:	
- Swap This with That	`s{`*ab*`}/^/`*ba*`/` where *a*, *b* are the two tags
- Put This in That	`s{`*a*`}/^/`*b*`/` puts the content of *a* into *b*
- Put This after That	`s{`*ab*`}/(.*)\r(.*)/`*b*`\n$2$1/` puts the content of *a* at the end of *b*
- Put This Before That	`s{`*ab*`}/(.*)\r(.*)/`*b*`\n$1$2/` puts the content of *a* at the beginning of *b*
Add/Remove Groups	
- Add	`s[Group:]{r}/(.*)/$d=$1; $d=~m!$ans[0]! ? $d : "$d, $ans[0]"/e`
- Remove	`s[Group:]{r}/(, )?$ans[0](, )?/$1 && $2 ? $1 : ""/eg`
New Last Played Date	`s[How many days ago?]{e}/.*/use POSIX; POSIX::strftime("%A, %B %e, %Y %l:%M:%S %p", localtime(time()-86400*$ans[0]))/e`
New Play Count	`s[New Play Count:]{p}/.*/$ans[0]/`
Append to Selected Artists Names	`s[Text to prepend:]{a}/^/a\n$ans[0]/`
Increase Play Count	`s{p}/(\d+)/"p\n" . (1+$1)/e`
Composer to 'Last, Rest-of-Name'	`s{C}/^(.*) (\S*)/C\r$2, $1/`
Artist to 'Last, First'	(see above)
Swap Album with Genre	`s{lg}/^/gl/`
Swap Song Name and Artist	`s{na}^/an/`
Composer-ize/Genre-ize	`s[Composer:]/.*/C\n$ans[0]/` or `s[Genre:]/.*/g\n$ans[0]/`
Rugged Good Looks	`s{aln}/(.*)/ ($d=$1) =~ s![]{,'"?&*()#\/\<> ]!!g; "aln\n$d" /e`
Replace Text	`s[Text to find:;Replace with:]{`*tags*`}/(.*)/ ($d=$1) =~ s!\Q$ans[0]]\E!$ans[1]!gi; "`*tags*`\n$d"`
Track Info Assist	`%a - %n`, `%a | %n`, `%C - %n`, etc.
Artist to Composer	`s{a}/^/C\n/`
Set Track Count	`s[Track Count:]/.*/T\n$ans[0]/`
Clean ID3 Tags	(covered by the other patterns. I wouldn't want to write this, but it's possible)
De-Genre	`s[Genre to eliminate:;Replacement Genre:]{g}/^$ans[0]$/$ans[1]/`
Filenames to Track Names	`s{f}/^/n\n/`
Filename to Comment Tag	`s{f}/^/c\n/`

That's 40 scripts! At the time of this writing, there are 58 scripts in the "Managing Track Info" section, so 70% of those scripts could have been spun off as one-liners with Track Parser.

The Parser
I've attempted to write the parser in a fairly generalized fashion, so it may be useful in other contexts as well. If you'd like to try it out, the source for the parser is contained in the "Parser (Song Names)" script. The important routines are `match`, `subst`, `split`, and `applyPattern`.

If you use my parser in any other script, please let me know, and be sure to give me credit somewhere.

Feedback
If you run into any problems that you couldn't have avoided by just doing a test run, or if you have any comments/suggestions, feel free to email me at danvk@rice.edu. Any ideas about what to call these scripts (yes, their names are pretty lame) definitely count as suggestions. If you come up with a new pattern that you find useful and think should be included by default, email it on over to me. The simplest way to copy a pattern is by examining it in the "…(Manager)" script. When the pattern appears, copy it to the clipboard.

Version History

1.3.1
Fixed pesky `Can't get every item of ""` bug introduced in iTunes 8.
Created a new home for Track Parser: http://code.google.com/p/trackparser/ where bugs/feature requests can be filed and commented on.
6/29/2009

1.3

Added the "no clobber" variants on several fields
Fixed all the uppercase fields, like "T" and "D". AppleScript really amazes me sometimes, the case in point here being the `consider case ... end considering` construct. "=" shouldn't return true when the two things aren't really equal!
Substitutions can now set more than just the name. This is an extremely powerful feature, as it lets you introduce new information into a match.
Substitutions can also take input from multiple sources. This allows you to do essentially anything that involves only specific track information (as opposed to "number of tracks selected," say).
Substitutions can now ask the user questions and use the response in the substitution. Mostly for use in spin-offs and saved patterns.
New fields: Played Count, BPM, Composer, Played Date, File Name (r/o), and Rating.
I really hope that the core pattern engine is feature-complete. Future changes are likely to be confined to improved error messages/handling.
Added the somewhat self-serving, self-gratifying "Imitating Other Scripts" section.
Added the ability to spin off patterns into standalone scripts.
Added the ability to save patterns in the "… (Clipboard)" script
Added a workaround for the iTunes 4.7 selection bug.
1/13/2005

1.2.1

Fixed those mysterious, annoying "Error -4690" messages that prevented people from using earlier versions.
The "…(Clipboard)" script now skips blank lines, or anything that it interprets as blank lines. This means that it runs as expected on text that you've copied out of a document in TextEdit with windows line endings.
7/27/2004

1.2

Added the "Favorites Patterns" screen, which appears by default when you run either script. This is a major shift in my approach to parsing, but I think it makes the scripts much easier to use.
Added the "Parser (Manager)" script to deal with favorites.
Changed the Read Me to reflect the above changes, focusing less on how to write expressions.
Added substitutions, mostly to allow the "Titlecase" pattern.
Fixed some quoting issues which prevented songs with apostrophes in them from being matched by regular expressions.
The "…(Clipboard)" script now gives you a sample line in the dialog.
The scripts (other than "…(Song Names)") now ask you where the song names script is if it's been renamed or moved.
7/11/2004

1.1

Added Regular Expression support via Perl
Added the "Test Run" functionality to the dialog box, which reduced the number of scripts to two.
The "… (Clipboard)" script now loads most of its code from the "…(Song Names)" script, which makes these nearly-identical scripts much easier to manage.
The Test Run now tells you which song it would be changing, instead of just the change.
You're now given the option to abort or continue if a line/song fails to match.
The Clipboard and Song Names scripts now remember the last patterns they successfully used.
6/14/2004

1.0

Initial Release
6/11/2004