Copy / Paste out of browser after ArtNet search:

On each page of the search results (or Print browser window – not the whole web page, just the search results) select all of the search results – ***including the number on the first result!!*** – then copy and paste into TextWrangler (or other text editor with Regular Expression find / replace)

Note spaces in the regular expressions below

PC Text Editor (e.g. Notepad++) Cleanup (PC lines end with \r\n)

Blank lines:

```
^[ \t]*\r\n
```
replace with nothing

Extra lines and artist name:

```
Full details\t\r\n(\d+)\t\r\n \t (.*)\r\n
```
replace with
```
Artist\t\2\t\1\r\n
```

Continued lines (don't contain tabs):

```
\r\n([^\t]*)\r\n
```
replace with *(space, not really underscore!!)*
```
_\1\r\n
```

Mac TextWrangler Cleanup (Mac lines end with \r)

Blank lines:

```
^[ \t]*\r
```
replace with nothing

Extra lines and artist name:

```
Full details\t\r(\d+)\t\r \t (.*)\r
```
replace with
```
Artist\t\2\t\1\r
```

Continued lines (don't contain tabs):

```
\r([^\t]*)\r
```
replace with (space, not really underscore)
```
_\1\r
```

Note: Beware of “Style of” artist lines: They may have different format that typical artist lines. We’ve found cases where these lines weren’t cleaned up properly using the above procedures. Either adjust the procedures to match the new pattern, or clean up those lines by hand to match the rest.

Regular expressions:

Regular expressions is a text pattern matching language. Most characters just match instances of themselves as you would expect in any find / replace search. There are some special characters, though, listed below like . ^ $ * +, which denote other things. Also, characters with \ slashes before them are either special characters, or it's a way to "escape" already special characters like . or \ to match that actual character in you text. This is just a few notes to get you started.

Characters

`\t`	`tab`
`\r`	`carriage return (end of line on Mac)`
`\r\n`	`return followed by newline (PC end of line combination)`
`\d`	`any digit (number character)`
`.`	`any character. Note: if you need to match an actual . you need to put a slash before it to "escape" it like \.`
`[ ]`	`match any character listed inside the brackets, including ranges. Note that inside of brackets, special characters like . just become themselves and aren't special`
`[0-9]`	`any digit character`
`[A-Za-z]`	`any upper or lowercase latin character`
`[^ ]`	`match any character except those listed inside brackets`

Quantities

`*`	`any number of repetitions of the previous character or group, including 0`
`+`	`one or more repetitions of the previous character or group`

Operators

`|`	`or operator (pipe character, shift-\) used inside parentheses to match either expression before or after`

Positional indicators

`^`	`beginning of a line`
`$`	`end of a line`

Capturing

`( )`	`on search, capture anything matching expression inside of parentheses to use later in the replacement expression`
`\1`	`on replace, use the 1st captured element`
`\2`	`on replace, use the 2nd captured element`

Google Refine Processing – Convert to one column per field format and add new columns based on extracting info from old ones with regex

- Create New Project and choose text file from regex cleanup
- Choose TSV as type of file
- Uncheck "Parse next 1 line(s) as column headers" so it won't use the first line of your real data as column headers
- *NEW:* Uncheck "Quotation marks are used to enclose cells containing column separators". Problems arise if unmatched double quotes are part of data. With that box checked, you're telling GR that, "I'll put double quotes around any part of the text where field delimiters (tabs in this case) should be interpreted as part of the text rather than as a field delimiter." So, it hits a double quote, and then it ignores any other tabs until it finds another double quote, or until the end of the file, whichever comes first.
- Click Create Project
- On column 3, choose Edit Cells -> Fill Down to copy unique identifier values to all rows of the data. GR needs this number to know which row in the eventual table each key/value pair goes with.
- On column 1 or 2, Transpose -> Columnize on key/value columns… with Column 1 as key and Column 2 as value (and don't choose anything in the third Notes column)
- Do any additional column creation with regex pattern matches as explained below.
- *NEW:* Export as TSV – TSV will be better if you've extracted prices or dates because it forces Excel to interpret the dates and the prices with commas in them as the correct data type instead of just as text. (Use File->Open, then from the drop-down at the bottom of the file open dialogue, choose Enable: All Files, because Excel doesn't recognize the .tsv extension)
- *NEW:* Google Refine allows you to save a series of operations and reapply them to a new data set https://code.google.com/p/google-refine/wiki/History

Sold for USD: On "Sold For" go to Edit Column -> Add column based on this column…
```
title: price_USD
expression: value.match(/(^|.*[ (])([0-9,]+) USD.*/)[1]
```

Note: Value.match() returns a list of captured matching items. The dollar amount is the 2nd piece I'm capturing with parentheses, so I need to use the [1] at the end of the expression to get the 2nd list element (in most programming languages you number list items starting with zero, not one).

Auction Date: on "Sale of" column, Edit Column -> Add column based on this column...
```
value.match(/(.*): [A-Za-z]+, (.*)\[(.*)/)[1]
```

Auction house: on "Sale of" column
```
value.match(/(.*?):.*/)[0]
```

Estimate range USD: from "Estimate" column. *Note: creating this intermediate column to make it easier to extract low and high estimate values below.*
```
value.match(/(^|.*[ (])([0-9,]+ - [0-9,]+) USD.*/)[1]
```

Estimate low USD: from intermediate "Estimate range" column. *Note: just replace [0] with [1] to get the high estimate from the range since this pattern is matching both dollar amounts.*
```
value.match(/([0-9,]+) - ([0-9,]+)/)[0]
```

Total number of Editions: 2nd number after a slash in "Editions" column. *Note: To grab the first number instead, replace [1] with [0] to return the 1st item in the list of captured matches.*
```
value.match(/(\d+)\/(\d+)/)[1]
```