Regular Expressions

The Download Overlay component includes the ability to select products with regular expressions. This article gives extensive examples of how to use regex effectively.

On this page...

The What and Why of Regular Expressions

A regular expression (often called a regex) is a powerful way to search text for matches. At the most basic level, it functions similarly to the "control /command + f" search function in most text editors and web browsers; any exact match with the "simple string" you enter will be returned. The advantage of regex is its specialized syntax, which allows for complex queries beyond simple 1:1 matching. This page will list the syntaxes available to you in the search form and walk through some examples of regex queries.

Regex is the filter of last resort!

It is likely that the filter you are attempting to apply can be found within the Download Overlay itself; you should start there.

Before you attempt to use advanced regex syntax in your search for files, you should try searching as you would in a browser or document. Often, a more complex query is not necessary to match your target filenames.

Common Regex Syntax

Not all regex syntax is valid for a filename search. The most useful syntaxes are included in the table below.

Syntax	Meaning
.	Matches any character
*	The preceding character appears zero or more times
+	The preceding character appears one or more times
$	Matches the end of a filename
[aeiou]	Matches any character in the listed set
[^XYZ]	Matches any character not in the listed set
[a-z0-9]	The set of characters can include a range or multiple ranges

If you want to use one of the special characters literally, you must 'escape' it. For example, if you actually wanted to include a period in your search query, you should enter "\."

An Important Distinction: * is not a wildcard

One of the most common mistakes in regex is using * as a generic wildcard character. As is shown in the example table below, the * character is always interpreted in conjunction with the preceding character.

The correct syntax for a wildcard is either ".*" or ".+", depending on whether the absence of a character should be included. See the last two columns of "regex input" for help clarifying this subtle point.

	Regex input			
Filename	**jwst*file**	**jwst.file**	**jwst.*file**	**jwst.+file**
jwstfile.fits	Match	No match	Match	No match
jwsttttfile.fits	Match	No match	Match	Match
jwstXfile.fits	No match	Match	Match	Match

jwst_abc123_file.fits	No match	No match	Match	Match

Example Searches

These examples use real filenames produced by the JWST mission. All text in the Regex Input column is "raw"; that is, you should enter it into the search bar exactly as you see it. Where special characters are used, they either take advantage of regex functionality or are part of the filename.

Regex Input	Match?	Notes
`rate`	**Matches** jw02783001001_04103_00001-seg001_mirimage_rate.fits jw02783001001_04103_00001-seg001_mirimage_rateints.fits **Does not match** jw02783001001_04103_00001-seg001_mirimage_uncal.fits	A regex search iterates through the filenames and returns matching strings. In this basic case, we've asked for all files that contain the phrase 'rate' somewhere within the name. It does not matter where this appears within the name; a simple string search is equivalent to a search in most text editors and internet browsers. The last example does not match because it does not have "rate" anywhere in its name.
`rate\.fits`	**Matches** jw02783001001_04103_00001-seg001_mirimage_rate.fits **Does not match** jw02783001001_04103_00001-seg001_mirimage_rateints.fits jwst_miri_filteroffset_0006.asdf	Although it's recommended to filter file types using the UI, it's also possible to filter using regex. The period is a special input that matches any character in our search string, so we escape it here using a backslash. It is good practice, but unnecessary in this case, to escape the period. There are no files with names like "rate9fits" that would cause an accidental match.

`fits$`	✓ **Matches** jwst_miri_flat_0789.fits jw02783001001_04103_00001-seg001_mirimage_rate.fits ! **Does not match** jwst_1077.pmap jwst_niriss_distortion_0032.asdf	Using the $ character, we can search for patterns at the end of a filename. This is particularly helpful in this example, where we look for filenames ending in fits. All other file types will be excluded.
`(fits|asdf)$`	✓ **Matches** jwst_miri_flat_0789.fits jwst_niriss_distortion_0032.asdf ! **Does not match** jwst_1077.pmap	We can combine $ with the "or" operator to great effect. In this query, we keep files ending in 'fits' or 'asdf', but exclude all others.
`(?<!fits|asdf)$`	✓ **Matches** jwst_1077.pmap ! **Does not match** jwst_miri_flat_0789.fits jwst_niriss_distortion_0032.asdf	This particular combination of special characters does the opposite of the last example: it excludes all fits and asdf files. You can chain together multiple, i.e. more than two, file types using the "or" operator. This uses an advanced regex operator called "look-behind". A word of caution that regex contains many such advanced/obscure operators. Getting them to work is only the first step; remembering *how* they work is even harder!

`seg00[123]`	✓ **Matches** jw02783001001_04103_00001-seg002_mirimage_calints.fits jw02783001001_04103_00001-seg003_mirimage_rate.fits ❗ **Does not match** jw02783001001_04103_00001-seg007_mirimage_x1dints.fits	Brackets limit the set of permissible characters. Our search in this example will find any file that contains 'seg00N', where N is either 1, 2, or 3. The last example does not match because '7' is not in the allowed set of characters.
`seg00[0-8]`	✓ **Matches** jw02783001001_04103_00001-seg008_mirimage_x1dints.fits ❗ **Does not match** jw02783001001_04103_00001-seg009_mirimage_x1dints.fits	Instead of specifying specific allowed characters, as we do in the above example, regex allows for ranges. In this case, we've allowed all numbers between (and inclusive of) 0 and 8. The last example does not match because '9' is not in the allowed set of characters.

`seg00[^9]`	✓ **Matches** jw02783001001_04103_00001-seg008_mirimage_x1dints.fits ❗ **Does not match** jw02783001001_04103_00001-seg009_mirimage_x1dints.fits	This search returns the same results as the previous, since "not 9" is equivalent to "the numbers 0 to 8" in this case. **Note:** In general, these searches are not equivalent. A filename containing "seg00B" would match *seg00[^9]*, since B is not 9. "seg00B" would not match with *seg00[0-8],* since B is not in the range 0-8. We can conveniently ignore this caveat for this search, as we know that "seg00B" is nonsense in a JWST filename.
`_[a-z]+ints`	✓ **Matches** jw02783001001_04103_00001-seg001_mirimage_calints.fits jw02783001001_04103_00001-seg004_mirimage_rateints.fits ❗ **Does not match** jw02783001001_04103_00001-seg009_mirimage_x1dints.fits	This search uses a wildcard character: '+', which means "the repeating character should show up at least once". We've used a clever trick here by combining it with "[a-z]", the set of all lowercase letters. In essence, we're asking: "Please find all strings that have an underscore, followed by any sequence of lowercase letters, followed by 'ints'." All of these example names end with _xxxints.fits. Our first two examples match because 'cal' and 'rate' are exclusively lowercase letters. The last example fails because 'x1d' contains a number.

For Further Reading...

- Regex101 is an excellent resource to "practice" these queries. The download overlay uses JavaScript syntax.