SEQUENCHER®

Tutorial for Windows and Macintosh

Advanced Handle Definition

Gene Codes Corporation

TCA GENE
AGT CODES

Gene Codes Corporation
525 Avis Drive, Ann Arbor, MI 48108 USA
1.800.497.4939 (USA) +1.734.769.7249 (elsewhere)
+1.734.769.7074 (fax)
www.genecodes.com gcinfo@genecodes.com

Advanced Handle Definition

Using Simple Delimiters 3

Writing Regular Expressions 4

Using the Expression is a delimiter Option 6

Using Advanced Regular Expressions 7

Getting More Help 9

Advanced Handle Definition

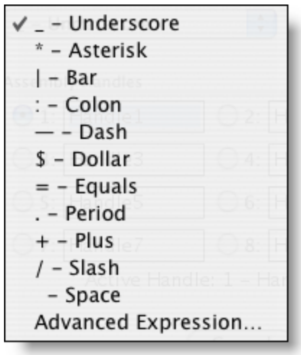
When you import your data into **Sequencher**, each imported sequence has a name. The Assemble by Name[1] function uses a portion of the sequence name that contains useful information about your sequencing reaction. We call each relevant portion of the sequence name an Assembly Handle. It might describe which sample was sequenced, which primer was used, or on which day the sequencing was performed. For any of your sequence names, you may define as many as eight different Assembly Handles.

Using Simple Delimiters

Properly defining the Assembly Handles is the key to the Assemble by Name function. Assemble by Name uses the active Assembly Handle to select the sequences that form the putative contig and to name the contig. For some sequence names this is easily done because the significant portions of the name are already separated one from the other by non-alphanumeric characters, such as an underscore or a hyphen. This separator, known as a delimiter, acts as a flag to separate the Assembly Handles.

Example 1:	Gel_Sample_Date_Oligo
Example 2:	Gel-Sample-Date-Oligo

Before you can **Assemble by Name**, you must set the rules that define the sequence name's relevant parts. In the simplest case, if your name structure follows the examples above, you may select a **Name Delimiter** from the **Assemble by Name Settings** dialog activated by clicking the **Name Settings** button on the **Assembly Parameters** dialog.



In the above examples, using the underscore as a delimiter, you generate four Assembly Handles: Gel, Sample, Date and Oligo. The descriptions for your Assembly Handles that you enter into the input fields allow you to more easily switch between active handles.

[1] More information regarding Assemble by Name can be found in the "Assemble by Name" tutorial.

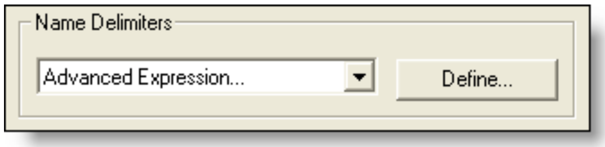
When you have a more complex naming scheme, a simple delimiter chosen from the list above may not properly segregate your sequence names into Assembly Handles. Your **Name Delimiter** may not be included in the drop down list, or it may consist of more than one character, or you may not have any delimiters in your name at all. Nevertheless, if you use a consistent naming scheme, you should also be able to break down your sequence name into **Assembly Handles** and thereby take advantage of **Assemble by Name.**

When you are not able to use the simple delimiters that **Sequencher** provides in the drop down menu to create your Assembly Handles, you may use regular expressions. Regular expressions are a set of symbols and rules that allow you to define matches in text. The advantage of a regular expression is that it is very flexible and relatively easy to learn, though reading a regular expression may take a little getting used to.

In **Sequencher**, you may use regular expressions to:

- Define the part of your sequence name that you want to use as a delimiter.
- Define the part(s) of the name that you want to use as Assembly Handles.

You access the regular expression editor by choosing Advanced Expression... from the Name Delimiters drop down menu. Then, click on the **Define...** button.



Writing Regular Expressions

Think of a regular expression as a set of instructions that allow you to identify matching text. Sometimes the match is very specific, as when you are searching for a given word or phrase. Sometimes the match may be more ambiguous, such as when you search for any phrase that starts with the letter "S".

For instance, if part of your regular expression includes the word:

Gel

wherever your sequence name contains the word "Gel", you will get a match. However, the match is case sensitive, so it will not match "gel" or "GEL".

So if you want to be able to match gel, Gel, or GEL you need to write a regular expression which directs **Sequencher** to recognize all of these cases. The bar, **|** , can be read as "or" in a regular expression. So, to match "gel", "Gel", or "GEL", type a **|** between the words.

Gel|gel|GEL

Regular expressions can also be built using wildcards. With these wildcards, you can specify not only a character or number, but how many occurrences of that character or number are present. These are listed in increasing specificity.

.	The period is the wildest of the wildcards. It represents any character.
[A-Za-z]	Represents any letter.
[A-Z]	Represents any capital letter.
[a-z]	Represents any lowercase letter.
\d and [0-9]	Both represent any digit.

The above special characters or phrases act as wildcards, and the following define the frequency of occurrences of specific characters or wildcards. Again, they are listed in increasing specificity.

*****	A character or special character followed by ***** matches *zero* or *more* instances of the preceding character.
+	A character or special character followed by **+** matches *one* or *more* instances of the preceding character.
?	A character or special character followed by **?** matches *zero* or *one* instances of the preceding character.
{N}	A character or special character followed by **{N}**, where "N" = a whole number, matches that many instances of the preceding character.

Other Special Characters:

[ ] These are very useful in regular expressions. The contents of square brackets describe a list of matching items. They work very differently from parentheses. For instance, if you type "Gel" in square brackets, you will match all items that contain a "G" or an "e" or an "l", regardless of order.

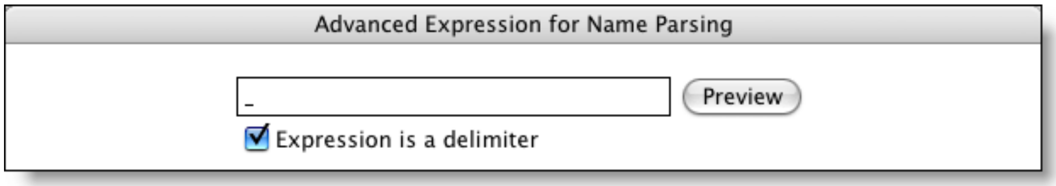
^ When you place a **^** in the square brackets, then your expression *excludes* the other items contained within the brackets.

\ Some characters have a special definition in regular expressions. If you need to use those characters, then precede the character with a forward slash. The \ is known as the escape character and it allows the character to be interpreted literally. It is also used to give a standard character a special regular expression definition, as it is with **\d** described above.

( ) Parentheses contain the portion of your regular expression that defines each separate Assembly Handle. The Assembly Handle must match whatever you type within the parentheses.

USING THE EXPRESSION IS A DELIMITER OPTION

Earlier we said that you could use the regular expression to define either the Assembly Handle or the delimiter. When you have a sequence that uses a delimiter that is not listed in the Name Delimiters drop down, the Advanced Expression option allows you to define a regular expression. Clicking on the **Define...** button to the right of the Name Delimiter drop down opens a new window where you can click on the **Expression is a delimiter** checkbox and enter an expression that describes the delimiter.

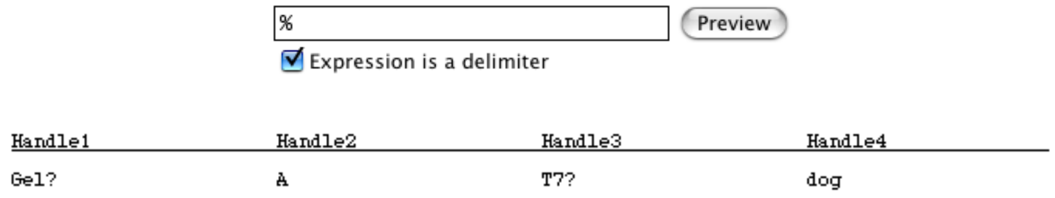

Advanced Expression for Name Parsing
_
Preview
Expression is a delimiter

The delimiter expressions are simpler to construct than the more advanced regular expressions described in the next section. If you have any sort of delimiter, try this option first.

A delimiter can be defined as a single character, or as a string of characters. You can also use special characters to create a more complex expression as a delimiter. The following four examples create expressions that define the boundaries of the Assembly Handles. They are all based on sequence: **Gel?%A%T7?%dog**.

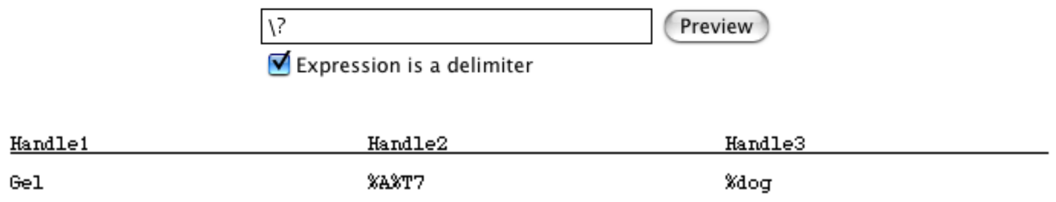
#	Delimiter Expression	Sequence Name	Handle1	Handle2	Handle3	Handle4	Handlle5	Handle6
1	**%**	Gel?**%**A**%**T7?**%**dog	Gel?	A	T7?	dog		
2	**\?**	Gel**?**%A%T7**?**%dog	Gel	%A%T7	%dog			
3	**\?%**	Gel**?%**A%T7**?%**dog	Gel	A%T7	dog			
4	**\?|%**	Gel**?%**A**%**T7**?%**dog	Gel		A	T7		dog

In example 1: the delimiter is a "**%**", which separates **Gel?%A%T7?%dog** into four handles. Hitting the **Preview** button after defining the expression shows *some* examples of the name parsing.



```
Handle1          Handle2          Handle3          Handle4
Gel?             A                T7?              dog
```

In example 2: **Gel?%A%T7?%dog**, the delimiter is "\**?**". The "\" is required because the "**?**" is a meta character, meaning that it has special meaning in regular expressions beyond its literal meaning. The "\" allows the "**?**" to be used as a literal character. Note that in this example, the "**%**" is included in the Assembly Handles because it is not recognized as a delimiter.

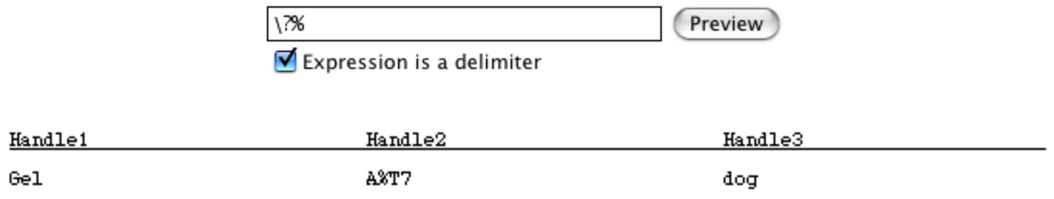


`\?`

☑ Expression is a delimiter

Handle1	Handle2	Handle3
Gel	%A%T7	%dog

In example 3: the delimiter is **"\?%"**. However, we again have to include the "\" to escape the "**?**" from its meta character meaning.

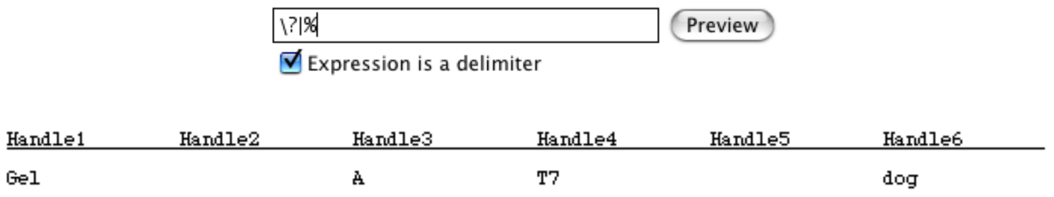


`\?%`

☑ Expression is a delimiter

Handle1	Handle2	Handle3
Gel	A%T7	dog

In example 4: **Gel?%A%T7?%dog**, the regular expression is "**?|%**", where the bar between the two delimiters is interpreted as an "or", so the delimiter is either the "**?**" or the "**%**". Wherever either of these characters occurs in the name, **Sequencher** creates a handle. Note that this results in two blank handles created for Handle 2 and Handle 5, which are defined by consecutive delimiters.



`\?|%`

☑ Expression is a delimiter

Handle1	Handle2	Handle3	Handle4	Handle5	Handle6
Gel		A	T7		dog

USING ADVANCED REGULAR EXPRESSIONS

When the **Expression is a delimiter** checkbox is not checked, your regular expression will include both the delimiters, if there are any, and the Assembly Handles. The regular expression that you write must describe the entire sequence name or it will not parse correctly. You may define from one to eight Assembly Handles, but if you have only one, it must differ from the sequence name by at least one character. In your regular expressions, what is in parentheses
( ) defines the Assembly Handles and what is outside of parentheses defines the delimiters.

(Assembly Handle 1**)**Delimiter**(**Assembly Handle 2**)**Delimiter**(**Assembly Handle 3**)**..., etc.

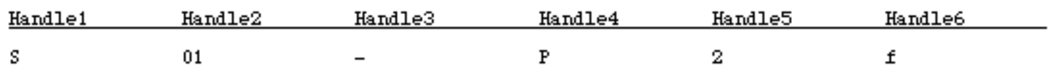
The sequence name **S01-P2f** will match the following regular expressions with increasing specificity. Those items that describe character **type** are in blue, character **frequency** are in green, and **literal** characters are in red.

.*	Matches any character zero or more times.
.+	Matches any character one or more times, so null values are not allowed.

.+..+	Two sets of characters of any length separated by a single character.
.+-.+	Two sets of characters of any length separated by a hyphen.
.{3}-.{3}	Two sets of three characters each separated by a hyphen.
[A-Za-z]+[0-9]+-[A-Za-z]+[0-9]+[A-Za-z]+	Some letters, followed by some numbers, followed by a hyphen, followed by some letters, some numbers, and some letters.
[A-Z][0-9]{2}-[A-Z][0-9][a-z]	A capital letter followed by two numbers, a hyphen, a capital letter, a number, and ending with a lower case letter.
S[0-9]{2}-P[0-9][r|f]	Capital "S" followed by two numbers then by "-P", followed by a number, followed by either lower case "r" or "f".
S01-P2f	The sequence name is not a valid handle definition.

To create Assembly Handles from your regular expression, enclose a portion or portions of the name in parentheses. Be careful to keep the two characters that describe type and frequency together. We will use the regular expression **[A-Z][0-9]{2}-[A-Z][0-9][a-z]** to demonstrate the effects of the location of the parentheses.

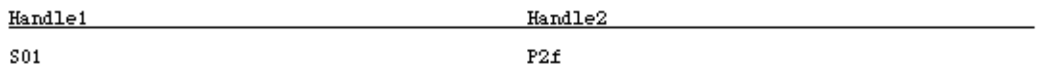
([A-Z])([0-9]{2})(-)([A-Z])([0-9])([a-z])



Handle1	Handle2	Handle3	Handle4	Handle5	Handle6
S	01	-	P	2	f

Here every portion of the regular expression is enclosed in parentheses, so concatenating the Assembly Handles recreates the sequence name; however, this needn't be the case.

([A-Z][0-9]{2})-([A-Z][0-9][a-z])

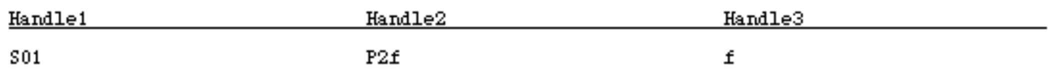


Handle1	Handle2
S01	P2f

The above regular expression segregates the sequence into two handles, but the hyphen is not included in either of the handles, because it is not enclosed in parentheses.

Sometimes you may want one portion of a sequence name to appear in two different Assembly Handles. The following example allows you to create two handles that both include the last lowercase letter.

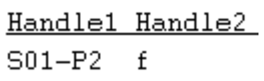
([A-Z][0-9]{2})-([A-Z][0-9]([a-z]))



Handle1	Handle2	Handle3
S01	P2f	f

One thing to keep in mind when using regular expressions is that one expression can have more power than another, and typically the expression to the left dominates those to the right. Look at the following examples:

(.*)(.+)　　**(.+)(.*)**　　**(.*)(.*)**　　**(.+)(.+)**



Handle1	Handle2
S01-P2	f

Remember that a "." represents any character. When an "*" follows, it allows for either zero or more instances of any character. When a "+" follows, it allows for *one* or more instances of any character. Whenever the regular expression has an "*" in the first Assembly Handle and a "+" in the second Assembly Handle, the second Assembly Handle contains only the required *one* character. If the regular expression describing the second Assembly Handle contains an "*", it may have a null value. In this case, the first Assembly Handle will equal the total sequence name.

Getting More Help

There are many internet resources that also describe how to use regular expressions. We suggest:
http://java.sun.com/docs/books/tutorial/essential/regex/index.html
Additionally, you have access to Gene Codes Technical Support. We will be glad to help you.