(Not all the macros are documented here, but only the generally
useful ones.)

== Add_Abbrev.bsh ==

This macro behaves like the built-in Expand Abbrev command, except
that it forces the display of the Add Abbrev Dialog even if an
expansion for the abbreviation before the cursor already exists.

== Align_on_Character.bsh ==

[This was based on the original Beautify Text Table by Corin]
[Langosch, and heavily modified to add many more features.   ]

USAGE

The macro operates on a selection, so you'll eventually need to
select a block of text before invoking it.

Put a command string in the '|' (vertical bar) register, in this
format:

/split regexp/delimiter/flags/column-spec/padding-char

(flags, column-spec, and padding-char are optional)

The '/' in the above line can actually be any non-alphabetic character
that will be used to mark the boundaries between the various parts of
the command string. Choose something that's not in your split regexp
or delimiter!

The 'flags' can include

B   Adds a delimter at the beginning of each line
E   Adds padding and a delimiter at the end of each line

```
    T   Trims whitespace from the whole line before writing it to
        the buffer. Useful if your delimiter has whitespace,
        but you only want it between columns.

  The column-spec allows you to left-, right-, or center-align the
  various columns of your input. Use a string of "L", "R", or "C", one
  for each column. If your input has fewer columns than the spec, the
  extras are ignored; and if your input has more columns, the default
  left-alignment is used.

  padding-char, if included, will be used instead of a space character
  to pad your columns.

  Example:

    /,\s+/ | /EBT/LLRC/*

  Parsed: <split char> = '/', split regexp = ",\\s+", delimiter = " | ",
          flags = "EBT", col-spec = "LLRC", padding-char="*"

  The defaults, used if the command-string register is empty,
  are "\t", " ", no flags, left-alignment, and spaces, respectively.


== Align_on_Comma.bsh ==

  A convenience macro for Align_on_Character, that aligns the columns
  of your select text by lining up the commas, and separating them
  with three spaces.


== Align_on_Prompted_Character.bsh ==

  Another convenience macro for Align_on_Character. It will prompt
  you for a keypress, and then use that punctuation character (regular
  letters or numbers won't work) to use as the delimeter of the columns
  in your selected text.


== Box_Selection.bsh ==

  This macro draws a box within the border of a rectangular selection
  using Unicode box drawing characters. It can draw a variety of box
  styles, depending on the value of the "JP.boxstyle" jEdit property.
  By default, it will draw single-lined boxes using Unicode
  characters. Set JP.boxstyle to "double" for double-line boxes,
  "rounded" for rounded-corners, or "ditaa" or "ascii" for an
  ASCII-only style compatible with the ditaa diagram tool
  (ditaa.sourceforge.net).
```

Examples (if these look messed up, download this .txt file and open
it in an editor that can properly display Unicode
characters in a monospaced font)

```
┌──────────────┐
│ BOXES!       │
└──────────────┘
                                ╭─────╮
                                │BUB- │
                                │ BLE │
                                ╰─────╯
```

```
╔═══════════════════╦═══════════════════════╗
║    FACT #         ║   HOW FASCINATING?    ║
╠═══════════════════╬═══════════════════════╣
║    232            ║      Not too much     ║
║    993            ║      A Little         ║
║      5            ║      Moderate         ║
║     32            ║      Incredibly       ║
╚═══════════════════╩═══════════════════════╝
```

Double-Walled Table

```
+--------------------+-----------------------+
|    FACT #          |   HOW FASCINATING?    |
+--------------------+-----------------------+
|    232             |      Not too much     |
|    993             |      A Little         |
|      5             |      Moderate         |
|     32             |      Incredibly       |
+--------------------+-----------------------+
```

ASCII Table

== Reflow_Selection.bsh ==

This is a big macro! If it's daunting, start by using the
Copy_as_Softflow.bsh helper.

It can reflow text in a variety of ways, and will save the reflowed
text to a register (recall that the special register '$' represents
the system clipboard); alternately, this macro treats the register
name "~" specially, and will replace the selection with the reflowed
text rather than saving it.

It pops up an input dialog to let the user choose what type of reflow
it should perform. Options are separated by spaces, and can be listed
in any order. Options may also be passed to the script, preventing
the popup, by loading a string into register 'R' of the format
"reflow:<options>".

Options:
---

s Soft reflow. This will join together lines that are not separated by a blank line into one logical paragraph. Such paragraphs are then separated from each other by a single newline. This method is good if you're going to import the resulting text to another program that performs its own line-wrapping, such as email, word-processors, etc. If <cols> is given, then lines that are shorter than <cols> will NOT be reformatted, and will be passed through into the output as-is.

h<cols> Hard reflow. This considers a group of lines that are not separated by a blank line as a logical paragraph, and joins them together into one block of text with <cols> columns. If <cols> isn't given, then the default of 72 is used.

t<s> Tabs. If this option is present, each paragraph will be indented by a tab character. For hard reflow, a tab is considered 8 characters. If <s> is given, then instead of a tab character <s> spaces will be used to lead paragraphs.

b<n> How many blank lines (<n>) separate logical paragraphs in the input text? The default is 1. Note that if you set this to 0, then every non-blank line is considered a paragraph.

n<i> Separates paragraphs in the output using <i> newlines. The default is to use 2 newlines (a blank line).

c Condense blank lines between paragraphs into one separator. Without this option, if, say, you pass b1 and n2, and two input paragraphs are separated by three blank lines, two additional newlines will be passed through to the output.

S<pd=> Suppress indentation on multiple blank lines. What this means is that if the input contains more blank lines than necessary (b<n>) to separate two paragraphs, the indentation for the second paragraph will be suppressed, like the first paragraph of a section in books.

   Additionally, if the argument is given as "Sp", a paragraph consisting entirely of punctuation (that is, \p{Punct} from the Pattern class) and whitespace, and which is less than 72 characters long, will cause the *next* paragraph to have its indentation suppressed. "Spd" will do the same, but will include digits ("\d") in the regexp.

   If you append an "=" to the suppression option (ex. "Sp="), then the paragraph that triggered suppression will have its own indentation suppressed as well.

```
  f     Recognize special formatting marks in the input text. Currently
        these include two special character sequences at the start of
        an input line that control the indentation for that paragraph,
        overriding the defaults:

            "|"   Do not indent this paragraph.
            "|>"  Indent this paragraph using a tab.

  e     Enables escape sequence processing in each paragraph's text.
        This is similar to escape sequences in Java string literals.
        Currently recognized escapes are: \t, \n, \\

  #   Enable hash-mark processing; with this option, a line consisting
      of only a hash mark will be treated as a blank line for
      paragraph delimitation, but will additionally insert a newline
      into the output. It can be used to make the layout of the source
      text more closely match the output. Note that using this option
      with b<n> where n>1 will lead to odd results.

  r<c>  Put the output into register <c>. Note that '$' is the system
        clipboard, the default.

  q     Quiet. Suppresses the completion message.
  ------------------------------------------------

  To prevent a section of text from being reformatted, enclose it
  between a "\begin{pre}" and "\end{pre}" on lines by themselves. The
  \begin{pre} and \end{pre} will not be included in the output.


== Copy_as_Softflow.bsh ==

  A convenience macro for Reflow_Selection, that copies the selection
  to the clipboard as soft-flowed paragraphs separated by a blank line.
  (see the "s" option for Reflow_Selection above).


== Copy_as_Tabflow.bsh ==

  A convenience macro for Reflow_Selection, that copies the selection
  to the clipboard as soft-flowed paragraphs with a tab at the start
  of each, not separated by any blank lines (see the 's', 't', & 'n'
  options for Reflow_Selection above).

== Unfill_Selection.bsh ==

  Uses Reflow_Selection to remove hard line breaks from paragraphs
  in the current selection (or the paragraph containing the caret,
  if there's no selection).
```

== Enumerate_Lines.bsh ==

Makes numbering lists in text documents much easier.

The macro operates on your current selection.

You can use it both to create new numbered lists:

```
Eggs                          1. Eggs
Cheese            ====>       2. Cheese
Milk                          3. Milk
Beefsteak                     4. Beefsteak
```

or to renumber already-enumerated lists if you insert or delete items.

```
1. Eggs                       1. Eggs
2. Cheese                     2. Cheese
2. Cereal               ===>  3. Cereal
   Wheeties, Cheerios            Wheeties, Cheerios
3. Milk                       4. Milk
4. Beefsteak                  5. Beefsteak
```

Notice that it only renumbered lines that already had a number, allowing you to use this macro on lists where certain items span more than one line. The only numerical value that it considers is that of the first item in your selection. Thus,

```
12. Eggs                      12. Eggs
3. Cheese         =====>      13. Cheese
6. Milk                       14. Milk
8. Beefsteak                  15. Beefsteak
```

== Evaluate_Preceding_Expression.bsh ==

Similar to jEdit's built-in "Evaluate Selection", save for two minor differences that make it quicker to invoke for simple expressions (like numerical calculations with money).

1. Instead of creating a selection and then evaluating it, this macro scans backward from the caret until it encounters a space, beginning of line, or a '$'. It treats the text between the caret and that point as the BeanShell expression to evaluate.

2. If the scan in the previous step stopped at a space or line start, then the text is replaced by the result of the evaluation as normal; but if it found a '$', then the expression is assumed to be a calculation involving money, and the result is treated as a Float and rounded to two decimal places. This avoids irksome floating point imprecisions in such calculations.

```
      ex.  Built-in Evaluation Selection (selecting "2.20+3.10"):
                   $2.20+3.10 ==> $5.300000000000001

           This macro:
                   $2.20+3.10 ==> $5.30
                              ^
                          caret here
```

```
== Fill_Paragraph.bsh ==

  (Note: Apache Commons Lang v3.4 or higher must be installed.)

  This is a fill paragraph macro that can fill certain types of
  paragraphs where the the average fill-paragraph command creates
  a jumble. Paragraphs like

      Tab-indented with subsequent lines
  not indented.

  - Lists where a bullet marks the first line
    but subsequent lines are indented.
       * And then sublists with different indentation
         levels with no blank lines in-between.

  To make the implementation of this macro easy, youhave to invoke it
  with the cursor *on the second or later* line of the paragraph.

  The macro looks at the indentation level of the current line, and
  uses that as the paragraph indentation; then it scans up until it
  finds a line with a different indentation, and this it considers the
  first line. The column at which the first line ends is used as the
  column to break later lines of the paragraph (unless "JP.wrapmargin"
  is set--see below). Then, scanning downward, the end of the
  paragraph is the last line that has the same indentation level as
  the rest of the paragraph.

  To alter the line-length behavior, set the jEdit property
  "JP.wrapmargin" to an integer column at which to wrap text. Setting
  it to a negative value will have the macro use the buffer's
  maxLineLen. To get the default behavior (i.e. using the length of
  the first line), unset the property or set JP.wrapmargin=0.


== Insert_Line_Below.bsh ==

  Very simple, but surprisingly useful. It inserts a blank line
  below the caret line, without moving the caret, like Emacs's
  C-o (open-line) command.
```

== Repeat_Character.bsh ==

  Inserts a character any number of times, like using the Action Bar,
  but quicker and without prompting for high repetitions.

  To use, create a "command string" in one of the below formats

    <char> x <count>   ex. "* x 15"
    <count><char>   ex. "60-"

  place the caret after the string and invoke this macro.

== Retain_Or_Remove_Lines.bsh ==

  This is a small modification of Jia Zhiming's (jiazhimi@yahoo.com.cn)
  macro of the same name. It uses Java's built-in regexp's and doesn't
  strip whitespace from retained lines.

== Sum-Count-Avg_Numbers.bsh ==

  Select a list of (float) numbers separated by commas, and this macro
  will insert a line into the buffer with the sum, count, and average
  of the numbers.

== Switch_Edit_Buffer.bsh ==

  Prompts for a number n, and then will switch the view to show the
  n-th open buffer. This is useful in combination with one of the
  plugins (I use Buffer Selector) that shows you your open buffers
  in order.