Mike Hahn – hahnbytes@gmail.com

Piqutary

Piqutary is a tool which lets you organize and share your image folders. It makes use of 2 open source third-party tools: an embedded web server called Jetty, and a text search engine called Lucene. Users must first launch the Piqutary web launcher and then point their web browsers to http://localhost:6886/. All image files are stored on the user's local hard drive. Piqutary makes use of 2 optional languages: a markup language (simplified HTML) called Piqutags, and a scripting language called Piqutalk, used to create 2-player board games.

Qunit Folders

A qunit is a queue of image files contained in a folder, and qunit folders can contain other qunit folders recursively. Newly added image files go to the head of the queue in each qunit. Every qunit contains a special queue which is a subset of the main queue of images. Both the main and the special queues support image reordering commands: head, tail, move left, move right. The head of the queue is the leftmost image in the queue. Every qunit has 2 yes/no flags: the H-flag and the X-flag. H stands for hidden (not shared publicly) and the X-flag warns users that its images are such that if the user is at work or sitting at a public computer, then proceed with caution. By default, qunits having an X-flag value of yes are hidden from the user. All lower level qunits of any hidden qunit or X-flagged qunit are also hidden or X-flagged, respectively.

Business Model

Piqutary no-name users pay no fees. Piqutary members pay a subscription fee of $10/year. All users can share qunits with other users by emailing links. All users can perform simple text-based searches, based on image captions, slide panels, and qunit names/descriptions. Only members can perform complex searches and bookmark qunits. All users can browse public (non-hidden) qunits of members without restrictions. All images contained in qunits of no-name users can be browsed, but any sub-qunits of a given no-name user's qunit are not displayed. All search results returned belonging to no-name users are stripped of all text: image captions, qunit names/descriptions, and user names. All of that same text information, including user names, is not displayed when browsing images in any qunit of a no-name user. No-name users can have many public qunits, but only one hidden qunit (which may contain zero or more numbered qunits).

Free	Members	Available Features
0	$10	Annual fee
yes	yes	Email links
yes	yes	Simple searches
–	yes	Complex searches
–	yes	Bookmark qunits
yes	yes	Browse qunits of members
–	yes	Sub-qunits displayed during no-name/member search
–	yes	Text meta-data displayed during no-name/member search
–	yes	Unlimited hidden qunits
yes	yes	Play/create Piqutalk games
–	yes	Upload Piqutalk games
–	yes	Post in forums, play in tournaments, hold player rankings

Commands

- **Enter** - down a level
- **Up Arrow** - up a level
- **Left/Right Arrow** - previous/next
    - qunit/screen/image/slide
- **Down Arrow** - toggle main/special
- **Ctrl+Down Arrow** - toggle slide mode
- **1** - first
- **0** - last
- **Shift+Up Arrow** - move to top
- **Shift+Left Arrow** - move left
- **Shift+Right Arrow** - move right
- **Shift+Down Arrow** - move to bottom
- **L** - like image: make it special
- **U** - undo L command
- **I** - insert image/qunit
- **D** - delete (slide mode)
- **Ctrl+D** - delete image/qunit
- **Ctrl+Z** - undo
- **Ctrl+X** - cut qunit
- **Ctrl+C** - copy qunit
- **Ctrl+V** - paste qunit
- **Ctrl+N** - new qunit
- **Ctrl+R** - rename qunit
- **J** - justify (left, center, right)
- **B** - bookmark qunit/access bookmarks
- **H** - hidden qunit on/off
- **X** - X-flag qunit on/off
- **Y** - sync qunit
- **S** - search
- **Q** - quit
- **F1** - cycle: menu/help/normal
- **F11** - fill screen
- **[x]** - close menu bar

Qunits Mode

Displays parent qunit name followed by an indented list of qunit names. Current qunit is highlighted (enclosed in square brackets). Qunit properties: name, description, img-flag, H-flag, X-flag. If img-flag is false, qunit contains no images, only other qunits. By convention, images stored in non-image qunits reside in a sub-qunit called "$".

Tiles Mode

Whenever the user is in Qunits Mode and presses Enter when an image qunit is highlighted, the current mode becomes Tiles Mode. The display is divided into 3 rows of equal height (or n rows where n > 1). Each row contains images. All portrait-mode images are of equal height but of varying widths. Every landscape-mode image is the same width as the height of the row which contains that image. All images are separated by a white, one-pixel gap (user may increase pixel count of gap, globally). Clicking on an image will display it in Image Mode. Pressing Up Arrow makes the current mode become Qunits Mode.

Image Mode

Single image is displayed, expanded by the maximum amount available on the user's display. Pressing L or U modifies special flag if needed and the current mode becomes Tiles Mode. Pressing Up Arrow makes the current mode become Tiles Mode. Pressing Enter enables editing of one-line caption, and/or toggling display of image captions, and/or toggling the filtering out of images which lack captions. If this image is accompanied by a link to a video, click on play to follow that link under a new browser tab. The image-view count of the user who is the image owner is incremented, if different from the user viewing the image.

Slide Mode

Press Ctrl+Down Arrow to toggle between Tiles Mode and Slide Mode. Slide Mode displays between 1 and 3 images per slide. Each slide tries to fill the entire display. Click on an image in slide mode to enter image mode, then press D to delete the image from the slide. In slide mode, press I to insert an image. The next time the I command is used in image mode, the image is inserted into the slide. Various image arrangements (side-by-side, stacked, or some combination) are used automatically, depending on the aspect ratios of the images on the slide. New slides can only be inserted at the head of the slide list. One of the panels on a given slide (which contains 1 to 3 panels) can contain Piqutags code instead of an image.

Qunit Sharing

Google Drive (or possibly Dropbox) will be used as the image sharing platform. Users can use Piqutary to keep track of their favorite image-sharing users and the names of their favorite image qunit names shared by those users.

Bookmark/Insert Commands

The Insert command displays the select-user web page. After selecting a user, the Local mode switch is off. Further Insert commands insert images/qunits into the current local qunit, and allow the user to change the current local qunit. The Quit command sets the Local mode switch back to on. The Bookmark command bookmarks the current qunit when Local mode is off (enabling bookmark parent list selection), and accesses the bookmark tree when Local mode is on.

Searching

Popularity is used in searches: how many times an image/qunit has been viewed, downloaded, or bookmarked. Searching is used to search for users and the images held by those users. Text searches involve qunit names/descriptions, image captions, and slide panels.

Image File Names

Newly added image files can have arbitrary file names. After the sync command is used, newly added image files (except for the least recently added image, which is manually appended with .1 when saved by the user) are renamed to $QNNNN.ext, where ext is a graphics file type such as PNG or JPEG and NNNN is a 4-digit number. Clock arithmetic is used, where 9999 corresponds to -1, 9998 corresponds to -2, and so on. Usually, the head of the queue is positive, and the tail of the queue is 0 or negative. No qunit can have more than 10,000 image files.

The sync command programmatically removes the .1 extension of the least recently added image. When the user goes to add the next batch of newly added images, and tries to save the same image file F that originally had the .1 extension, it will already exist in the current qunit. This will let the user know when to stop adding images. The next time the sync command is performed, image file F is renamed to have the same $QNNNN format as all of the older image files in the current qunit.

About Me

I am Mike Hahn, the founder of Piqutary.com. I was previously employed at Brooklyn Computer Systems as a Delphi Programmer and a Technical Writer (I worked there between 1996 and 2013). At the end of 2014 I quit my job as a volunteer tutor at Fred Victor on Tuesday afternoons, where for 5 years I taught math, computers, and literacy. I'm now a volunteer math/computer tutor at West Neighbourhood House. My hobbies are reading quora.com questions/answers and the news at cbc.ca. About twice a year I get together with my sister Cathy who lives in Victoria. She comes here or I go out there usually in the summer. A few months prior to starting my Piqutary project I used to lie on the couch a lot, not being very active. Now I'm busy most of the time. I visit my brother Dave once a month or so and I also visit my friends Main and Steph once or twice a month.

Contact Info

Mike Hahn
Founder, Piqutary.com
515-2495 Dundas St. West
Toronto, ON M6P 1X4

Country: Canada
Phone: 416-533-4417
Email: hahnbytes (AT) gmail (DOT) com
Web: www.hahnbytes.com

Database

- User
  - username: 50
  - firstname: 50
  - lastname: 50
  - email: 255
  - password: 40
  - phone: 20
  - balance: money
  - qunid
  - mnuid
  - isactive
  - ismem
  - ispaused
  - startdate
  - memdate
  - expiredate
  - pausedate
  - returndate
  - leavedate
- Qunit
  - usrid
  - parentid
  - nextid
  - childid
  - name: 50
  - desc: 255
  - isimg
  - ishidden
  - isxflag
  - pos
  - endpos
  - favid
  - favendid
  - sldid
  - sldendid
- Slide
  - qunid
  - nextid
- Menu (bookmarks)
  - usrid
  - parentid
  - nextid
  - childid
  - qnnid
  - text: 50
- Qunitnode
  - mnuid
  - qunid
  - nextid
  - favid
- Favnode
  - qunid
  - imgid
  - nextid
- Image
  - qunid
  - qpos: short
  - width: short
  - height: short
  - ext: byte
  - isfav
  - iscap
  - isaltcap (tags)
- Caption
  - imgid
  - iscap
  - text: 80
- Imgpanel
  - sldid
  - imgid
- Textpanel
  - sldid
  - text: 32K

Implementation Steps

1. Read Murach's Java Servlets and JSP book
2. Implement local mode
3. Implement server-side code
4. Write basic Piqutags design specs
5. Implement Piqutalk 0.1, console-based
   - Token parsing and building program tree has already been implemented
6. Finish Piqutalk 1.0, console-based
7. Write advanced Piqutags design specs
8. Implement PQTG-to-HTML converter
9. Implement monospace mode
10. Implement rich-text mode

11. Integrate Piqutalk with Piqutags (monospace/rich-text modes)
12. Implement PQTK-to-JS converter
13. Recruit GitHub open source coders/testers
14. Implement Jabbler: web-based Scrabble game, user vs. robot
    1. Jabbler is currently console-based Java Scrabble game
15. Implement web-based chess and backgammon:
    1. Play against robot which makes random but legal moves
16. Implement Jabbler: web-based, 2-player
17. Implement monospace mode, dual user
18. Implement rich-text mode, dual user
19. Implement Piqutalk SDK
    1. WYSIWYG Piqutags editor
    2. Piqutalk code editor
    3. Convert Jabbler from Java to Piqutalk
    4. WYSIWYG board/piece editor
    5. Codeless prototyping system
20. Design website
21. Launch website
22. Beta test Piqutary
23. Advertise using Google AdWords
24. Accept credit card payments
25. Hire Java programmer with expertise in making websites scalable

Piqutags

Piqutags is a simplified markup language used to replace HTML. Arbitrary Piqutags code can be embedded in the Piqutalk echo statement. Piqutags syntax, where asterisk (*) means repetition, is defined as follows:

- Tags:
    - [tag]
    - [tag: body]
    - [tag (fld val)*: body]
- Body:
    - text
    - [: text]*
    - [(fld val)*: text]*
- Call Piqutalk code:
    - [expr: <expr>]
    - [exec: <stmt>...]
    - [pqtk: <path>]

Monospace Mode

In monospace mode, all body text rendered to the screens of end-users is in a mono-spaced, typewriter-style font. Every character takes up 2 square cells: an upper cell and a lower cell. Superscripts and subscripts are handled by employing a vertical offset of one square cell. Header text is also mono-spaced, and each character takes up 2 oversized square cells.

Additional Formatting

The grid of characters can be subdivided into panels, which can themselves be subdivided into more panels, and so on. Any panel can contain zero or more text boxes, which may overlap each other. Vertical grid lines each take up one square cell per row of square cells. Horizontal grid lines are displayed in the same pixel row as underscore characters. Any row of square cells containing a horizontal grid line which is 2 pixels wide is taller by exactly one pixel. The following bracket characters: ( ) [ ] { } can be oriented vertically or horizontally, taking up a single column or row of at least 2 square cells, respectively. Widgets such as check boxes, radio buttons, and combo box arrows take up 4 square cells (2 by 2). Images, animations, and diagrams are contained in canvas objects, which can appear anywhere panels can appear.

Rich-Text Mode

In rich-text mode, a given header or paragraph of body text can consist of a single variable-width font. Paragraphs have before/after spacing, left/right indent, and line spacing (single, double, 1.5, etc.). Panels have margins on all 4 sides. In both rich-text and monospace modes, text is rendered to the HTML5 canvas object. Some features like form fields and submit buttons use hidden HTML.

Piqutalk Scripts

Members are allowed to upload Piqutalk game scripts (2-player board games), and all users can download scripts created by the members. Piqutalk is a new open source web programming language used to create apps which run locally, in your browser. Piqutalk is accompanied by Piqutags, a replacement for HTML. All or most Piqutary profits are distributed to the app writers in proportion to the user-minutes accumulated by each app. Piqutalk includes functionality which detects inactivity, so idle user-minutes don't count. Members can post in forums, play in tournaments, and hold player rankings.

Piqutalk

Piqutalk is a Python dialect in which all operators precede their operands, and parentheses are used for all grouping (except string literals, which are delimited with double quotes). Piqutalk code is often accompanied by Piqutags screen definition files. Piqutags is similar to HTML, except open tags begin with an open square bracket and a keyword, and the closing tag is simply a close square bracket. Any text enclosed in a tag is preceded by a colon. File extensions include .PQTK (Piqutalk) and .PQTG (Piqutags).

Special Characters

- `()` grouping
- `-` used in identifiers
- `;` end of stmt.
- `:` dot operator
- `"` string delimiter
- `\` escape char.
- `#` comment
- Extra:
    - `_` used in identifiers
    - `$` string prefix char.
    - `*` public switch
    - `{}` block comment

Version 0.1

- No inheritance
- No interfaces
- No IDE
- No rich text

Differences from Python

- Parentheses, not whitespace
- Integration with Piqutags
- Operators come before their operands
- Information hiding (public/private)
- Single, not multiple inheritance
- Adds interfaces ("scool" defs.)
- Drops iterators and generators
- Adds lambdas
- Adds quote and list-compile functions, treating code as data

Grammar Notation

- Non-terminal symbol: <symbol>
- Optional text in brackets: [*text*]
- Repeats zero or more times: [*text*]…
- Repeats one or more times: <symbol>…
- Pipe separates alternatives: *opt1* **|** *opt2*
- Comments in *italics*

Keyboard Aid

This optional feature enables hyphens, open parentheses, and close parentheses to be entered by typing semicolons, commas, and periods, respectively. When enabled, keyboard aid can be temporarily suppressed by using the Ctrl key in conjunction with typing semicolons, commas, and periods (no character substitution takes place). By convention, hyphens are used to separate words in multi-word identifiers, but semicolons are easier to type than hyphens. Similarly, commas and periods are easier to type than parentheses. Typing semicolon converts previous hyphen to a semicolon, and previous semicolon to a hyphen (use the Ctrl key to override this behaviour). Typing semicolon after close parenthesis simply inserts semicolon. The close delim switch automatically inserts a closing parenthesis/double quote when the open delimiter is inserted.

Piqutalk Grammar

White space occurs between tokens (parentheses and semicolons need no adjacent white space, also any semicolon before a close parenthesis may be omitted):

```
<source file>:
•   [<use>] [ * <vars>] [<vars>] [ do <block>]
    [<def>]… [<class>]… [ do <block>]

<use>:
    use ( <import-semi>... )

<import-semi>:
    <import-stmt> ;

<import stmt>:
    import <module>
    import ( <module>… )
    from <rel module> import <mod list>
    from <rel module> import all

<module>:
    <name>
    ( <name> as <name> )
    ( : <name>… [ as <name>] )

<mod list>:
    <id as>
    ( <id as>… )

<id as>:
    <mod id>
    ( <mod id> as <name> )

<mod id>:
    <mod name>
    <class name>
    <func name>
    <var name>

<rel module>:
    ( : [<num>] [<name>]... )
    <name>   // ?

<class>:
•   ( [ * ] <cls typ><name> [<base class>] [<does>]
    [ * <vars>] [<vars>] <def>… )
•   ( [ * ] scool <name> [<does>] [<const list>]
    [<def hdr>]... )
•   ( [ * ] enum <name><elist> )

<cls typ>:
    class
    abclass

<does>:
    does ( <scool name>... )

<scool name>:
<base class>:
    <name>
    ( : <name><name>… )

<const list>:
    const ( <const pair>... )

<const pair>:
    ( <name><const expr> )

<def hdr>:
    ( <defun><name> ( [<parms>] ) [<dec>] )

<def>:
•   ( [ * ] <defun><name> ( [<parms>] ) [<vars>]
    [<dec>] do <block> )

<defun>:
    def
    abdef

<vars>:
    var ( <id>... )

<parms>:
    <parm>... [ ( * <id> ) ] [ ( ** <id> ) ]

<parm>:
    <id>
    ( tuple <id>... )
    ( <set op><id><expr> )
    ( <set op> ( tuple <id>... ) <expr> )

<dec>:
    decor <call expr>...

<block>:
    ( [<stmt-semi>]… )

<stmt-semi>:
    <stmt> ;
```

```
<jump stmt>:
      <continue stmt>
      <break stmt>
      <return stmt>

<pjump stmt>:
      return <expr>
      ** <raise stmt>

<raise stmt>:
      raise [<expr> [ from <expr>] ]

<stmt>:
      <open stmt>
      <closed stmt>

<open stmt>:
      <if stmt>
      <while stmt>
      <for stmt>
      ** <try stmt>
      <pjump stmt>
      <pcall stmt>
      <asst stmt>
      <del stmt>

<closed stmt>:
      <jump stmt>
      <call stmt>
      <print stmt>
      <lstg tag>

<call expr>:
•     ( <name> [<arg list>] )
•     ( : <obj expr> [<colon expr>]…
      ( <method name> [<arg list>] ))
•     ( call <expr> [<arg list>] )

<call stmt>:
      ( <name> [<arg list>] )

<pcall stmt>:
•     : <obj expr> [<colon expr>]…
      ( <method name> [<arg list>] )
•     call <expr> [<arg list>]

<colon expr>:
      <name>
      ( <name> [<arg list>] )

<arg list>:
      [<expr>]... [ ( <set op><id><expr> ) ]...
```

```
<asst stmt>:
      <asst op><target expr><expr>
      <set op> ( tuple <target expr>... ) <expr>

<asst op>:
      set | addset | minusset | mpyset | divset |
      idivset | modset |
      shlset | shrset | shruset |
      andbset | xorbset | orbset |
      andset | xorset | orset |
      = | += | -= | *= | /= |
      //= | %= |
      <<= | >>= | >>>= |
      &= | ^= | '|=' |
      &&= | ^^= | '||='

<set op>:
      set | =

<target expr>:
      <name>
      ( : <name> [<colon expr>]… <name> )
      ( slice <arr><expr> [<expr>] )
      ( slice <arr><expr> all )

<arr>:         // string or array/lyst
      <name>
      <expr>

<obj expr>:
      <name>
      <call stmt>

<if stmt>:
•     if <expr> do <block> [ elif <expr> do <block>]… [
      else <block>]

<while stmt>:
      while <expr> do <block>
      do <block> while <expr>

<for stmt>:
      for <name> in <expr> do <block>

<try stmt>:
•     try <block> <except clause>… [ else <block>]
      [ finally <block>]
•     try <block> finally <block>

<except clause>:
      except <name> [ as <name>] do <block>

<return stmt>:
      return

<break stmt>:
      break
```

```
<continue stmt>:
    continue

<del stmt>:
    del <expr>

<paren stmt>:
    ( <open stmt> )
    <closed stmt>

<qblock>:
    ( quote [<paren stmt>]... )

<expr>:
    <keyword const>
    <literal>
    <name>
    ( <unary op><expr> )
    ( <bin op><expr><expr> )
    ( <multi op><expr><expr>… )
    ( <quest><expr><expr><expr> )
    <lambda>
    ( quote <expr>... )
    <renum expr>
    <tuple expr>
    <lyst expr>
    <dict expr>
    <bitarray expr>
    <string expr>
    <bytezero expr>
    <bytes expr>
    <target expr>
    <obj expr>
    <cast>

<quest>:
    quest | ?

<unary op>:
    minus | notbitz | not |
    - | ~ | !

<bin op>:
    <arith op>
    <comparison op>
    <shift op>
    <bitwise op>
    <boolean op>

<arith op>:
    div | idiv | mod | mpy | add | minus |
    / | // | % | * | + | -

<comparison op>:
    ge | le | gt | lt | eq | ne | is | in |
    >= | <= | > | < | == | !=
```

```
<shift op>:
    shl | shr | shru |
    << | >> | >>>
```

Note: some operators delimited with single quotes for clarity (quotes omitted in source code)

```
<bitwise op>:
    andbitz | xorbitz | orbitz |
    & | ^ | '|'

<boolean op>:
    and | xor | or |
    && | ^^ |  '||'

<multi op>:
    mpy | add | strdo | strcat |
    and | xor | andbitz | xorbitz |
    or | orbitz |
    * | + | % | + |
    && | ^^ | & | ^ |
    '||' | '|'

<const expr>:
    <literal>
    <keyword const>

<literal>:
    <num lit>
    <str lit>
    <bytes lit>

<tuple expr>:
    ( tuple <expr>… )

<lyst expr>:
    ( lyst [<expr>]… )

<dict expr>:
    ( dict [<pair>]… )

<bitarray expr>:
    ( bitarray <enum name> [<elist>] )
    ( bitarray <enum name><idpair>... )

<elist>:
    <id>...
    <intpair>...
    <chpair>...

<intpair>
    // integer constant
    <int const>
    ( <int const><int const> )
```

```
<chpair>
    // one-char. string
    <char lit>
    ( <char lit><char lit> )

<idpair>
    <idt>
    ( <id><id> )

<pair>:
    // expr1 is a string
    ( <expr1><expr2> )
    ( <str lit><expr> )

<renum expr>
    ( renumize <expr><ren id>... )
    ( renumize <expr><ren int>... )
    ( renumize <expr><ren ch>... )

<ren id>:
    ( 0 <id> )
    ( 1 <id> )
    ( 1 <id><id> )

<ren int>:
<ren ch>:
    // expr is <dec int> | <char lit>
    ( 0 <expr> )
    ( 0 <expr><expr> )
    ( 1 <expr> )
    ( 1 <expr><expr> )

<cast>:
    ( cast <type><expr> )

<print stmt>:      // built-in func
    ( print <expr>… )
    ( println [<expr>]… )
    ( echo <expr>… )

<lambda>:
    ( lambda ( [<id>]... ) <expr> )
    ( lambda ( [<id>]... ) do <block> )
    ( lambda ( [<id>]... ) do <qblock> )
    // must pass qblock thru compile func
```

No white space allowed between tokens, for rest of Piqutalk Grammar

```
<white space>:
    <white token>...

<white token>:
    <white char>
    <line-comment>
    <blk-comment>

<line-comment>:
    # [<char>]... <new-line>

<blk-comment>:
    { [<char>]... }

<white char>:
    <space> | <tab> | <new-line>

<name>:
•   [<underscore>]… <letter> [<alnum>]…
    [<hyphen-alnum>]… [<underscore>]…

<hyphen-alnum>:
    <hyphen><alnum>…

<alnum>:
    <letter>
    <digit>
```

In plain English, names begin and end with zero or more underscores. In between is a letter followed by zero or more alphanumeric characters. Names may also contain hyphens, where each hyphen is preceded and succeeded by an alphanumeric character.

```
<num lit>:
    <dec int>
    <long int>
    <oct int>
    <hex int>
    <bin int>
    <float>

<dec int>:
    [<hyphen>] 0
    [<hyphen>] <any digit except 0> [<digit>]…

<long int>:
    <dec int> L
```

```
<float>:
    <dec int><fraction> [<exponent>]
    <dec int><exponent>

<fraction>:
    <dot> [<digit>]…

<exponent>:
    <e> [<sign>] <digit>…
<e>:
    e | E

<sign>:
    + | -

<keyword const>:
    none
    true
    false

<oct int>:
    0o <octal digit>…

<hex int>:
    0x <hex digit>…
    0X <hex digit>…

<bin int>:
    0b <zero or one>…
    0B <zero or one>…

<octal digit>:
    0 | 1 | 2 | 3 | 4 | 5 | 6 | 7

<hex digit>:
    <digit>
    A | B | C | D | E | F
    a | b | c | d | e | f

<string lit>:
    [ $ <str prefix>] <short long>

<str prefix>:
    r | u | R | U

<short long>:
    " [<short item>]... "
    " " " [<long item>]... " " "

<short item>:
    <short char>
    <escaped str char>

<long item>:
    <long char>
    <escaped str char>
```

```
<short char>:
    any source char. except "\", newline, or
    end quote

<long char>:
    any source char. except "\"

<bytes lit>:
    $ <byte prefix><shortb longb>

<byte prefix>:   // any case/order
    b | br

<shortb longb>:
    " [<shortb item>]... "
    " " " [<longb item>]... " " "

<shortb item>:
    <shortb char>
    <escaped char>

<longb item>:
    <longb char>
    <escaped char>

<shortb char>:
    any ASCII char. except "\", newline, or
    end quote

<longb char>:
    any ASCII char. except "\"

<escaped char>:
    \newline
        ignore "\", newline chars.
    \\  backslash
    \"  double quote
    \}  close brace
    \a  bell
    \b  backspace
    \f  formfeed
    \n  new line
    \r  carriage return
    \t  tab
    \v  vertical tab
    \ooo    octal value = ooo
    \xhh    hex value = hh

<escaped str char>:
    <escaped char>
    \N{name}    Unicode char. = name
    \uxxxx      hex value (16-bit) = xxxx
```