Rimu Tips

HTML is valid Rimu

You are free to mix any valid HTML into a Rimu document — no special syntax or relearning is required (in this sense Rimu can be thought of as an HTML superset).

Converting Rimu to PDF

First convert Rimu markup to HTML and then convert the HTML to PDF.

Google's Chrome web browser has an option to print HTML to a PDF file, other browsers have PDF add-ons that perform the same function.

Alternatively use an HTML to PDF conversion program such as HTMLDOC.

Extend Rimu with Macros

Macros provide a simple, consistent mechanism for extending the Rimu markup vocabulary. For example:

Rimu source

```
{sidebar} = '==== $1
.[style="padding-left: 10px; border-left: 4px solid silver; margin-bottom: 1em;"
{info} = '<span class="label label-info">$1</span>'
{heads-up} = '{info|Heads up!}'
{note} = '{info|Note} <mark>$1</mark>'
{playground} = '<http://rimumarkup.org/rimuplayground.html|Rimu
Playground>'

{sidebar|Fun with Macros}
..
{heads-up} See the {playground} for documentation and hands-on
experimentation.

{note|This document is not yet finished.}
..
```

Rendered HTML

<h4>Fun with Macros</h4>
Heads up! See the Rimu Playground for documentation and hands-on experimentation.
Note This document is not yet finished.

- The `{sidebar}` macro shows how you can inject CSS styles into a *Division Block*, but a better way of styling block elements is to use scoped CSS.
- Note how macro definitions can span multiple lines and can contain any valid Rimu markup (including HTML and blank lines).

Syntax highlighting

Highlight.js is a nice syntax highlighter that makes it trivial to add syntax highlighting to Rimu Code blocks and Indented paragraphs. The *rimuc* tool `--styled` option includes Highlight.js if you set the `{highlightjs}` macro to a non-blank value. It does this by linking to CSS and JavaScript files on the *Highlight.js* website.

Highlight.js automatically detects the language in Code Blocks and Indented Paragraphs. You can also set the language explicitly by adding a code block class, for example:

Rimu source

```
{highlightjs} = 'yes'

.javascript
--
function die(message) {
  if (message) {
    console.error(message);
  }
  else {
    console.error('fatal error');
  }
  process.exit(1);
}
--
```

Rendered HTML

```
function die(message) {
  if (message) {
    console.error(message);
  }
  else {
    console.error('fatal error');
  }
  process.exit(1);
}
```

Set the `no-highlight` class to disable syntax highlighting for a code block. Adding the `no-highlight` class to all code blocks using *Delimited Block Definitions* effectively turns off auto-highlighting unless the language class is explicitly set:

```
|code| = '<pre class="no-highlight"><code>|</code></pre>'
|indented| = '<pre class="no-highlight"><code>|</code></pre>'
```

Syntax shared with Markdown

The following subset of *Rimu* elements is compatible with Markdown:

Headers
: Headers using # character header IDs.

Code blocks
: Indented paragraphs (at least four spaces) render like Markdown code blocks.

Text formatting
: `_emphasis_` and `` `code` `` quotes.

Lists
: - Bullet lists with `-` and `*` IDs.
  - Numbered lists with explicit numbering.

URLs and Email addresses
: Using the `<url>` and `<email>` syntaxes.

The Rimu *README* uses this subset for compatibility with *GitHub* and *npmjs.org* README formats.

Escaping Rimu syntax

If you have text that you don't want interpreted as a Rimu element then you can render it literally (escape it) by prefixing the element with a backslash character.

Passing macro values into Rimu documents

You can do this by prepending text containing macro definitions to Rimu source before rendering the source. The *rimuc* command has a `--prepend` option that can be used to do this.

Alternatively you could also put macro definitions in a separate file and specify it as the first Rimu source file in the *rimuc* command.

The following example sets the HTML *title* in the `kotlin-notes.html` output document:

```
rimuc --styled --prepend "{title}='Kotlin Notes'" kotlin-notes.rmu
```

Syntax checker

The Vim editor syntax highlighter is a big help when writing (and reading) Rimu markup.

The Rimu markup looks ok but the generated HTML is wrong

Using an editor with Rimu syntax highlighting support (see previous topic) makes it much easier to spot syntax errors.

Display contiguous block images vertically

Unless they are separated by other block elements, multiple block image elements flow inline like text (this is because the HTML `img` element is an "inline block" element), you can flow them vertically by enclosing them in a *Division Block* (HTML `div` element). For example:

Rimu source

```
<image:http://www.w3.org/Icons/w3c_home.png|W3C logo>
..
.#big-logo [width="150"]
<image:http://www.w3.org/Icons/w3c_home.png|W3C logo>
..
```

Rendered HTML

Entering non-ASCII characters

A number of non-ASCII Unicode characters are commonly used for punctuation e.g. em dash, ellipsis and quotation characters. You can enter these characters (or any Unicode character) as HTML character entities or you can enter them literally.

For example, to enter an ellipsis you could either enter `&hellip;` or you could enter the actual … character directly (alternatively you could define a *Replacement* for the ellipsis, see the Example .rimurc file below). The mechanism for entering Unicode characters directly will be dictated your editor:

Vim
: Enter *Ctrl+V* followed by the the character *u* followed by the hexidecimal character code e.g. to enter an em dash enter *Ctrl+V* followed by *u2014*

Linux console input
: Enter *Ctrl+Shift+U* followed by the hexadecimal character code followed by the *Enter* key.

Common Unicode characters:

```
Character     HTML entity    Hexadecimal code
---------     -----------    ----------------
—             &mdash;        2014
…             &hellip;       2026
“             &ldquo;        201c
”             &rdquo;        201d
→             &rightarrow;   2192
```

Here's a full list of HTML5 character entities.

Whitespace and Underscores in URLs

Whitespace characters are illegal in URL paths, use *%20* in place of spaces, for example:

```
<file:///home/srackham/books/War%20and%20Peace.pdf|War and Peace>
```

Underscore characters are legal in URL paths but they may be interpreted as Rimu emphasis quotes, you can use *%5F* in place of underscores or escape them with backslashes.

Use Replacement and Quote definitions sparingly

Adding new Replacement and Quote definitions changes the Rimu syntax which can produce unexpected results. It can also make your Rimu source non-portable and less readable.

Example .rimurc file

Here is an example `.rimurc` file containing *Macro*, *Quote* and *Replacement* definitions (refer to the Rimu documentation under *Tools*):

```
/*
```

```
  ~/.rimurc file containing custom Rimu Markup definitions used by
  rimuc command.
*/

/*
  Replacements
*/
// Don't match HTML comment tags.
/(^|[^!])--(?!>)/ = '$1&mdash;'
/\.{3}/ = '&hellip;'
/``/ = '&ldquo;'
/''/ = '&rdquo;'
// Don't match HTML close comment tag.
/(^|[^-])->/ = '$1&rightarrow;'
/<-/ = '&leftarrow;'
/\bTODO\b/ = '<b style="color: red; background-color: yellow;">TODO</b>'
/^(NOTE|IMPORTANT|WARNING|TIP):/ = '<b>$1</b>:'
// Encodes (most) raw HTTP URLs as links.
/(^|[^<])((?:http|https):\/\/[^\s"']*[^\s"',.;?)])/ = '$1<a href="$2">$2</a>'
// Alternative (typing friendly) emphasis, works for most cases but
// can't contain quotes.
/\B'\b(.+?)\b'\B/g = '<em>$1</em>'

/*
  Macros
*/
{mark} = '<mark>$1</mark>'
{del} = '<del>$1</del>'
{ins} = '<ins>$1</ins>'
{sup} = '<sup>$1</sup>'
{sub} = '<sub>$1</sub>'

/*
  Quotes
*/
~ = '<del>|</del>'

/*
  Highlight.js
*/
// Turn off auto-hightlighting.
|code| = '<pre class="no-highlight"><code>|</code></pre>'
|indented| = '<pre class="no-highlight"><code>|</code></pre>'
```

Extending the Vim syntax highlighter

If you've added custom replacements and quotes you can highlight them in Vim by creating a custom `~/.vim/after/syntax/rimu.vim` syntax file. Vim loads this file after the distributed `~/.vim/syntax/rimu.vim` file. The following example `~/.vim/after/syntax/rimu.vim` syntax file highlights the admonitions replacement and the strike-through quote definitions from the previous example `.rimurc` file:

```
" ~/.vim/after/syntax/rimu.vim
"
" Custom Vim highlighting for custom syntax defined in ~/.rimurc
"

" Admonishments.
syn match rimuAdmonition /^\([A-Z]\+\):/ containedin=ALLBUT,rimuComment,rimuCodeBlock
hi def link rimuAdmonition Special

" ~ quote.
syn match rimuSpanDeleted /\\\@<!\~[ \t\n]\@!\(.\|\n\(\s*\n\)\@!\)\{-1,}[\\
```

```
\t\n]\@<!\~/ contains=rimuSpanEntity
hi def link rimuSpanDeleted Todo

" Raw HTTP URLs as links.
syn match rimuSpanRawURL /[\\<]\@<!\(http\|https\):\/\/[^\s"']*[^\s"',.;?)]/
hi def link rimuSpanRawURL Title
```

Understanding substitution priorities

Inline elements are processed in the following order (first to last):

1. Macros
2. Quotes
3. Replacements
4. Special characters

Substitutions in multi-line block elements can be controlled using *block-options* in the *Block Attributes* element.

Using Macro, Replacement and Quote definitions in Safe Mode

Macro, *Replacement* and *Quote* definitions are not processed in Safe Mode. To apply Macro, Replacement or Quote definitions to Rimu markup processed in Safe Mode you need to load the definitions with a separate API call, for example:

```
Rimu.render(trusted_rimu_definitions); // safeMode is off by default.
var html = Rimu.render(untrusted_rimu_markup, {safeMode: 1});
```

Modifying built-in Quotes and Replacements

You can (with caveats) redefine built-in *Quotes* and *Replacements*, here are a couple of examples:

```
_ = '<em class="emphasis">|</em>'
/\\?<(\S+@[\w\.\-]+)>/g = '<a class="mailto" href="mailto:$1">$1</a>'
```

Refer to the Rimu source files quotes.ts and replacements.ts.

Mathematical formulae using MathJax

You can include math formulas written in LaTeX or MathML using MathJax. The `rimuc` command `--styled` option will include MathJax support if the `{mathjax}` macro is defined. Examples:

Rimu source

```
{mathjax} = 'yes'

.-macros
A LaTeX inline formula: \(\sum_{i=0}^n i^2 = \frac{n^2+n}m{2}\)

A LaTeX block formula:

.-macros
\[\sum_{i=0}^n i^2 = \frac{n^2+n}m{2}\]

A MathML inline formula:
```

```
<math>
  <mi>E</mi><mo>=</mo><mrow><mi>m</mi><msup><mi>c</mi><mn>2</mn></msup></mrow>
</math>

A MathML block formula:

<math display="block">
  <mi>E</mi><mo>=</mo><mrow><mi>m</mi><msup><mi>c</mi><mn>2</mn></msup></mrow>
</math>


A LaTeX inline formula: \(\sum_{i=0}^n i^2 = \frac{n^2+n}m{2}\)

A LaTeX block formula:

\[\sum_{i=0}^n i^2 = \frac{n^2+n}m{2}\]
```

Rendered HTML

A MathML inline formula: E=mc2

A MathML block formula:

E=mc2

NOTE: Macro expansion has been disabled in LaTeX formulas to stop `{i=0}` and `{2}` from being mistaken for Rimu macro invocations.

See the latest MathJax documentation to learn how to use MathJax.

Conditional Inclusion

Inclusion macro invocations can be used to conditionally include and exclude lines of source text:

- If-then-else logic:

  ```
  {macro=} Included if macro is undefined or value is blank.
  {macro!} Included if macro is defined and value not blank.
  ```

- Case statement logic:

  ```
  {macro=one} Included if macro value is 'one'.
  {macro=two} Included if macro value is 'two'.
  {macro=\d+} Included if macro value is a number.
  ```

- Any block element, except for Line Block definitions, can be prefixed with an *Inlusion* macro For example:

  ```
  {highlightjs!}<script src="highlight.min.js"></script>
  ```

Macros also provide a handy way to comment out text within block elements that support macro expansion (for example paragraphs and HTML blocks). Here is an example paragraph that uses this technique:

```
Lobortis lacus vestibulum sem.
{other-text!} This line is excluded from the output.
Mauris vitae ipsum. {other-text|This parameter text
is excluded} Donec dui.
```

So long as `other-text` is undefined or blank the text is excluded — set it to `$1` and the text will be included. Note that parameter text can extend over multiple lines.

Tables

Create tables using the HTML table syntax (Rimu has no special syntax for tables).

You can use Rimu markup in table cells, just remember that HTML blocks must start at the left margin and end with a blank line.

The `rimuc --styled` option includes Bootstrap so you can use Bootstrap tables classes to create nicely formatted tables. For example:

Rimu source

Rendered HTML

Lorem ipsum dolor. Sit amet venenatis. Erat nulla arcu. Lorem luctus sem. Mauris vitae ipsum. Donec dui ac ridiculus quisque proin. Lobortis lacus vestibulum sem.

- Lorem ipsum dolor sit
- lorem dolor fusce nec.
    - Sit pretium qui

Normal paragraph

Lorem ipsum dolor. Sit `amet venenatis`. Erat nulla arcu. Lorem luctus sem. *Mauris vitae* ipsum. Donec dui ac ridiculus quisque proin. Lobortis lacus vestibulum sem.

Indented paragraph

```
Neque a massa. Porttitor consectetuer commodo. Nunc nulla tempor
tempor enim ornare. Mi nam id. Ornare mauris tempor. Pede mauris
sed.  Scelerisque feugiat massa alias.
```

- Ac curabitur el
  - Platea a
    1.
    2.
  - Pellente
- Augue et dui malesuad

  ```
  If a list item is
  paragraph is inclu
  ```

Macro meta-programming

You can define macros which, when invoked, generate other macros. A good example of this is the generation of document section headers, links and table of contents entries from a section identifier and title:

Rimu source

```
// Macro to generate section link, header and TOC entry macros.
// $1 = section ID, $2 = section title.
{section-definition} = '
{$1} = '<#$1|$2>' \
{$1-header} = '<<#$1>>
== $2' \
{$1-toc} = '{$1} \' \
'
// Generate section macros.
{section-definition|section1|Section One}
{section-definition|section2|Section Two}

==== Table of Contents
{section1-toc}
{section2-toc}

// Section One section.
{section1-header}
Link to {section2}.

// Section Two section.
{section2-header}
Link to {section1}.
```

Table of Contents

Rendered HTML

Section One

Link to Section Two.

Section Two

Link to Section One.

NOTE: The backslash character is used to continue macro definition lines that end with a single-quote.

Macro tips

- Use the `&vert;` character entity to display a `|` character inside a macro parameter value.
- Closing block delimiters must be explicit and cannot be sourced from a macro invocation (because Rimu elements are recognized before macro expansion).
- The contents of a macro can be rendered verbatim in a code block by enabling macro expansion. For example:

  ```
  .+macros
  --
  {example-42}
  --
  ```

  The contents of `{example-42}` are not subject to macro expansion.

- Definition elements cannot be prefixed with an Inclusion macro (see Conditional Inclusion).
- Macro invocations in macro definition values are expanded when the macro is declared, not when it is invoked. You can defer evaluation until the macro is invoked by escaping macro invocations or (in the case of multi-line definitions) by using the `-macros` *Block Attributes* option. For example this Rimu markup:

  ```
  {m1} = 'foo'
  {m2} = '{m1} \{m1}'
  {m2}

  {m1} = 'bar'
  {m2}
  ```

  Generates:

  ```
  <p>foo foo</p>
  <p>foo bar</p>
  ```

Use scoped CSS to generate custom styled elements

HTML5's scoped CSS attribute allows you to define CSS outside the document header. This allows you to include CSS in the body of a Rimu document which is very handy for tweaking the default CSS and for styling Rimu block elements. Examples:

The above *verse* and *sidebar* CSS styles are automatically included when you use the *rimuc* tool `--styled` option.