Built-In Helpers

The `if` block helper

You can use the `if` helper to conditionally render a block. If its argument returns `false`, `undefined`, `null`, `""`, `0`, or `[]`, Handlebars will not render the block.

```
<div class="entry">
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{/if}}
</div>
```

when used with an empty (`{}`) context, `author` will be `undefined`, resulting in:

```
<div class="entry">
</div>
```

When using a block expression, you can specify a template section to run if the expression returns a falsy value. The section, marked by `{{else}}` is called an "else section".

```
<div class="entry">
  {{#if author}}
    <h1>{{firstName}} {{lastName}}</h1>
  {{else}}
    <h1>Unknown Author</h1>
  {{/if}}
</div>
```

The `unless` block helper

You can use the `unless` helper as the inverse of the `if` helper. Its block will be rendered if the expression returns a falsy value.

```
<div class="entry">
  {{#unless license}}
  <h3 class="warning">WARNING: This entry does not have a license!</h3>
  {{/unless}}
</div>
```

If looking up `license` under the current context returns a falsy value, Handlebars will render the warning. Otherwise, it will render nothing.

The `each` block helper

You can iterate over a list using the built-in `each` helper. Inside the block, you can use `this` to reference the element being iterated over.

```
<ul class="people_list">
```

```
<ul class="people_list">
  {{#each people}}
    <li>{{this}}</li>
  {{/each}}
</ul>
```

when used with this context:

```
{
  people: [
    "Yehuda Katz",
    "Alan Johnson",
    "Charles Jolley"
  ]
}
```

will result in:

```
<ul class="people_list">
  <li>Yehuda Katz</li>
  <li>Alan Johnson</li>
  <li>Charles Jolley</li>
</ul>
```

You can use the `this` expression in any context to reference the current context.

You can optionally provide an `{{else}}` section which will display only when the list is empty.

```
{{#each paragraphs}}
  <p>{{this}}</p>
{{else}}
  <p class="empty">No content</p>
{{/each}}
```

When looping through items in `each`, you can optionally reference the current loop index via

`{{@index}}`

```
{{#each array}}
  {{@index}}: {{this}}
{{/each}}
```

Additionally for object iteration, `{{@key}}` references the current key name:

```
{{#each object}}
  {{@key}}: {{this}}
{{/each}}
```

The first and last steps of iteration are noted via the `@first` and `@last` variables when iterating over an array. When iterating over an object only the `@first` is available.

Nested `each` blocks may access the interation variables via depth based paths. To access the parent index, for example, `{{@../index}}` can be used.

The `each` helper also supports block parameters, allowing for named references anywhere in the block.

```
{{#each array as |value key|}}
  {{#each child as |childValue childKey|}}
    {{key}} - {{childKey}}. {{childValue}}
  {{/each}}
{{/each}}
```

Will create a `key` and `value` variable that children may access without the need for depthed variable references. In the example above, `{{key}}` is identical to `{{@../key}}` but in many cases is more readable.

The `with` Block Helper

Normally, Handlebars templates are evaluated against the context passed into the compiled method.

```
var source   = "<p>{{lastName}}, {{firstName}}</p>";
var template = Handlebars.compile(source);
template({firstName: "Alan", lastName: "Johnson"});
```

results in

```
<p>Johnson, Alan</p>
```

You can shift the context for a section of a template by using the built-in `with` block helper.

```
<div class="entry">
  <h1>{{title}}</h1>

  {{#with author}}
  <h2>By {{firstName}} {{lastName}}</h2>
  {{/with}}
</div>
```

when used with this context:

```
{
  title: "My first post!",
  author: {
    firstName: "Charles",
    lastName: "Jolley"
  }
}
```

will result in:

```
<div class="entry">
  <h1>My first post!</h1>

  <h2>By Charles Jolley</h2>
</div>
```

`with` can also be used with block parameters to define known references in the current block. The example above can be converted to

```
<div class="entry">
  <h1>{{title}}</h1>

  {{#with author as |myAuthor|}}
  <h2>By {{myAuthor.firstName}} {{myAuthor.lastName}}</h2>
  {{/with}}
</div>
```

Which allows for complex templates to potentially provide clearer code than `../` depthed references allow for.

You can optionally provide an `{{else}}` section which will display only when the passed value is empty.

```
{{#with author}}
  <p>{{name}}</p>
{{else}}
  <p class="empty">No content</p>
{{/with}}
```

The `lookup` helper

The `lookup` helper allows for dynamic parameter resolution using Handlebars variables. This is useful for resolving values for array indexes.

```
{{#each bar}}
  {{lookup ../foo @index}}
{{/each}}
```

The `log` block helper

The `log` helper allows for logging of context state while executing a template.

```
{{log "Look at me!"}}
```

Delegates to `Handlebars.logger.log` which may be overriden to perform custom logging.

Any number of arguments may be passed to this method and all will be forwarded to the logger.

```
{{log "This is logged" foo "And so is this"}}
```

The log level may be set using the `level` hash parameter. Supported values are `debug`, `info`, `warn`, and `error`. When omitted, `info` is the default value,

```
{{log "Log!" level="error"}}
```

Logging is conditional based on the level and to value set in `Handlebars.logger.level`, which defaults to `info`. All log statements at or above the current level will be output.

The `blockHelperMissing` helper

Implicitly called when a helper can not be directly resolved in the environment's helpers hash.

```
{{#foo}}{{/foo}}
```

will call this helper with the resolved value of `foo` on the current context and the `options.name` field set to `"foo"`. For instances where there is no registered helper named `foo`.

This may be overriden by users that wish to change the behavior of block evaluation. For example

```
Handlebars.registerHelper('blockHelperMissing', function(context, options) {
  throw new Handlebars.Exception('Only if or each is allowed');
});
```

could be used to prevent the use of mustache-style block evaluation in favor of the more efficent `if` and `each` helpers.

The `helperMissing` helper

Internal helper that is called when a potential helper expression was not found in either the environment helpers or the current context. For cases where both are run, this is run prior to the `blockHelperMissing` helper.

```
{{foo}}
{{foo bar}}
{{#foo}}{{/foo}}
```

Will each call this helper, passing any arguments that would have been otherwise passed to a helper of the same name. This helper is not called when using `knownHelpersOnly` mode.

This may be overriden by applications. To force the existence of the field, the following may be used:

```
Handlebars.registerHelper('helperMissing', function(/* [args, ] options */) {
  var options = arguments[arguments.length - 1];
  throw new Handlebars.Exception('Unknown field: ' + options.name);
});
```

Found a documentation issue? Tell us!