handlebars

Handlebars allows for template reuse through partials. Partials are normal Handlebars templates that may be called directly by other templates.

Basic Partials

In order to use a partial, it must be registered via `Handlebars.registerPartial`.

```
Handlebars.registerPartial('myPartial', '{{name}}')
```

This call will register the `myPartial` partial. Partials may be precompiled and the precompiled template passed into the second parameter.

Calling the partial is done through the partial call syntax:

```
{{> myPartial }}
```

Will render the partial named `myPartial`. When the partial executes, it will be run under the current execution context.

Dynamic Partials

It's possible to dynamically select the partial to be executed by using sub expression syntax.

```
{{> (whichPartial) }}
```

Will evaluate `whichPartial` and then render the partial whose name is returned by this function.

Subexpressions do not resolve variables so `whichPartial` must be a function. If a simple variable has the partial name, it's possible to resolve it via the `lookup` helper.

```
{{> (lookup . 'myVariable') }}
```

Partial Contexts

It's possible to execute partials on a custom context by passing in the context to the partial call.

```
{{> myPartial myOtherContext }}
```

Partial Parameters

Custom data can be passed to partials through hash parameters.

```
{{> myPartial parameter=value }}
```

Will set `parameter` to `value` when the partial runs

This is particularly useful for exposing data from parent contexts to the partial:

```
{{> myPartial name=../name }}
```

Partial Blocks

The normal behavior when attempting to render a partial that is not found is for the implementation to throw an error. If failover is desired instead, partials may be called using the block syntax.

```
{{#> myPartial }}
  Failover content
{{/myPartial}}
```

Which will render `Failover content` if the `myPartial` partial is not registered.

This block syntax may also be used to pass templates to the partial, which can be executed by the specially named partial, `@partial-block`. A template of

```
{{#> layout }}
  My Content
{{/layout}}
```

with the `layout` partial containing

```
Site Content
{{> @partial-block }}
```

Would render

```
Site Content
```

```
My Content
```

When called in this manner, the block will execute under the context of the partial at the time of the call. Depthed paths and block parameters operate relative to the partial block rather than the partial template.

```
{{#each children as |child|}}
  {{#> childEntry}}
    {{child.value}}
  {{/childEntry}}
{{/each}}
```

Will render `child.value` from this template, not the partial.

Inline Partials

Templates may define block scoped partials via the `inline` decorator.

```
{{#*inline "myPartial"}}
  My Content
{{/inline}}
{{#each children}}
  {{> myPartial}}
{{/each}}
```

Which will render the `myPartial` partial for each child.

Each inline partial is available to the current block and all children, including execution of other partials. This allows for layout templates and similar functionality:

```
{{#> layout}}
  {{#*inline "nav"}}

    My Nav
```

```
  {{/inline}}
  {{#*inline "content"}}
    My Content
  {{/inline}}
{{/layout}}
```

Where the `layout` partial may be:

```
<div class="nav">
  {{> nav}}
</div>
<div class="content">
  {{> content}}
</div>
```