Progress API

Progress Overview

The Fluid Progress component provides a usable and accessible linear progress display for use on its own or with other Fluid components. It was originally designed for the Fluid Uploader but was built to be highly flexible -- customizable for use in any context. It has a very simple API but achieves its flexibility through a number of configuration options which at first may not seem intuitive.

Fluid Progress does not *currently* come with a default presentation. We hope that through our design examples you can come up with your own designs that integrate well with your application or component. Fluid Progress is currently used in the Uploader and in the Fluid Project wiki to provide component progress indicators.

New in v1.3

The Progress component now provides the option to the integrator to switch between `aria-valuenow` and `aria-valuetext`. See the fluid:Options description below for more information.

The Progress component now fires events when progress begins and hides.

Status

This component is in Preview status

On This Page

See Also

- Progress
- Tutorial - Progress
- Progress QA Test Plan
- Fluid Component API

Still need help?

Join the infusion-users mailing list and ask your questions there.

Creating a Progress bar

To instantiate a new Progress component on your page:

```
var myProgressBar = fluid.progress(container, options);
```

Returns: The Progress component object.

Note: the initial state of a progress element is assumed to be hidden with the minimum amount of progress.

Parameters

`container:` a CSS-based selector, single-element jQuery object, or DOM element that identifies the root DOM node of the Progress markup.

`options:` an optional data structure that configures the Progress component, as described in the fluid:Options section of this page.

Supported Events

The Progress component fires the following events (for more information about events in the Fluid Framework, see Events for Component Users):

Event	Type	Description	Parameters	Parameter Description
New in v1.3: `onProgressBegin`	default	This event fires once the progress display has appeared. Note that this is *not* a 'preventable' event.	none	
New in v1.3: `afterProgressHidden`	default	This event fires once the progress display has been removed after progress is complete.	none	

To add listeners to the firers one can use two approaches:
1) Using the `listeners` option to the component creator function (see #Options below for more information):

```
//onProgressBegin:
var myProgress = fluid.progress("#progress-container", {
    listeners: {
        onProgressBegin: myProgressShow //callback function
    }
});
//afterProgressHidden:
var myProgress = fluid.progress("#progress-container", {
    listeners: {
        afterProgressHidden: myProgressHide //callback function
    }
});
```

2) Programmatically:

```
//onProgressBegin:
myProgress.events.onProgressBegin.addListener(myProgressShow);
//afterProgressHidden:
myProgress.events.afterProgressHidden.addListener(myProgressHide);
```

Methods

Method	Description	Parameters
`show(animation)`	Shows the element defined by the *`displayElement`* selector using either the default *`showAnimation`* object parameters or an animation passed in.	**`animation:`** see below the description of animate object used in the *`showAnimation`* option.
`hide(delay, animation)`	Hides the element defined by the *`displayElement`* selector using either the default *`hideAnimation`* object parameters or an animation passed in	**`delay:`** see below the description of the *`delay`* option **`animation:`** see below the description of animate object used in the *`hideAnimation`* option
`update (percentage, labelValue, animationForShow)`	Updates the *`indicator`* element with a new percentage complete, updates the visible *`label`*, and *`ariaElement`*, shows the *`displayElement`* if it is currently hidden using either the default *`showAnimation`* object parameters or an animation passed in.	**`percentage:`** an integer value between 0 and 100 indicating the current progress (numbers greater than 100 are interpreted as 100) **`labelValue:`** a string to display in the *`label`* element **`animationForShow:`** see below the description of animate object used in the fluid:*`showAnimation`* option.

`refreshView()`	Resets the position and size of the *`indicator`* element based on the current size and position of the *`progressBar`* element. Useful for liquid layouts or events that change the location or size of the *`progressBar`*.	*none*

Options

Name	Description	Values	Default

`selectors`	Javascript object containing selectors for various fragments of the Progress markup	The object must contain a subset of the following keys:
		`displayElement`
		`progressBar`
		`indicator`
		`label`
		`ariaElement`
		See fluid:selectors below for requirement details
			`selectors: {`
			`    displayElement: ".flc-progress",`
			`    progressBar: ".flc-progress-bar",`
			`    indicator: ".flc-progress-indicator",`
			`    label: ".flc-progress-label",`
			`    ariaElement: ".flc-progress-bar"`
			`}`
			See #Selectors below for requirement details

`strings`
New in v1.3
	The strings that will be used by the component. This is where localization is handled
	
	`ariaBusyText`
	String to inject into the *ariaElement* during *update()* while the percentage is less than 100%.
	The *%percentComplete* token is replaced with a number indicating the current percent complete.
	New in v1.3
	If `ariaBusyText` is provided, the `aria-valuetext` attribute will be updated using this test. Otherwise, the value of the `aria-valuenow` attribute will be used for `aria-valuetext`.
	
	`ariaDoneText`
	String to inject into the *ariaElement* during *update()* when the percentage complete is 100%.
		The object must contain a subset of the following keys:
		`ariaBusyText`
		`ariaDoneText`
			`strings: {`
			`    ariaBusyText: "Progress is %percentComplete percent complete",`
			`    ariaDoneText: "Progress is complete."`
			`}`

`showAnimation`	JavaScript object that defines the default animation for displaying the Progress *`displayElement`*.
	
	The default `showAnimation` can be overridden at run-time by passing an animation object in with the `progress.show()` or `progress.update()` methods.
		The structure of the object mirrors and gets mapped to the parameters of jQuery's animate method.
		
		NOTE: The `callback` property is deprecated in v1.3
			`showAnimation: {`
			`    params: {`
			`        opacity: "show"`
			`    },`
			`    duration: "slow",`
			`    callback: null`
			`}, // equivalent of $().fadeIn("slow")`

`hideAnimation`	JavaScript object that defines the default animation for hiding the Progress *`displayElement`*.
	
	The default `hideAnimation` can be overridden at run-time by passing an animation object in with the `progress.hide()` method.
		The structure of the object mirrors and gets mapped to the parameters of jQuery's animate method.
		
		NOTE: The `callback` property is deprecated in v1.3
			`hideAnimation: {`
			`    params: {`
			`        opacity: "hide"`
			`    },`
			`    duration: "slow",`
			`    callback: null`
			`}, // equivalent of $().fadeOut("slow")`

`listeners` New in v1.3	JavaScript object containing listeners to be attached to the supported events.	Keys in the object are event names, values are functions or arrays of functions.	See #Supported Events for more information.

`minWidth`	Integer which specifies the minimum width for the progress indicator element	pixels	`minWidth: 5`
`delay`	delay before hiding the progress after the Progress.hide() method. a delay allows the user to register the completion of progress before hiding the progress bar.	millies (milliseconds)	`delay: 0`
`speed`	Integer representing the speed for the Progress animations. The default is very fast because you want the animation to keep up with the actual speed of the action.		`speed: 200`
`animate`	String indicating which directions of progress get animated. In most cases you only want to animate forward.	"forward", "backward", and "both". Any other value is interpreted to mean don't animate at all	`animate: "forward"`
`initallyHidden`	Boolean indicating whether to hide the *`displayElement`* when initializing Progress.	boolean	`initiallyHidden: true`
~~`updatePosition`~~	Deprecated: Replaced by the *refreshView()* method	boolean	`updatePosition: false`
~~`ariaBusyText`~~	Moved in v1.3: Moved into `strings` block	string	`ariaBusyText: "Progress is %percentComplete percent complete"`
~~`ariaDoneText`~~	Moved in v1.3: Moved into `strings` block	string	`ariaDoneText: "Progress is complete."`

Selectors

Selectors are used to indicate which elements in DOM should behave as the different Progress elements.
The value for the option is itself a Javascript object containing name/value pairs:

```
selectors: {
    selector1Name: "selector 1 string",
    selector2Name: "selector 2 string",
      ...
}
```

Each selector has a default, as defined below.

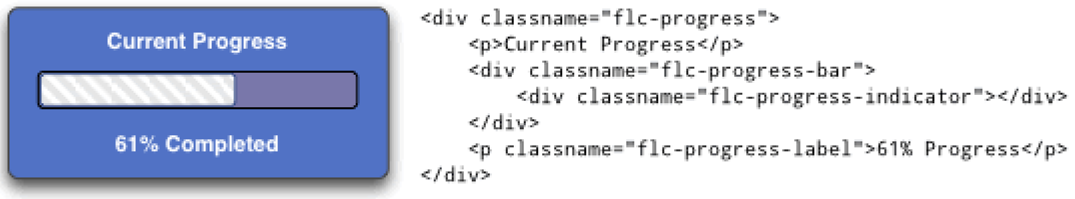
Selector name	Description	Default
`displayElement`	[fluid:required] the element that gets displayed when progress is displayed, could be the indicator or bar or some larger outer wrapper as in an overlay or dialog effect	".flc-progress"
`progressBar`	[fluid:required] The container for a file row progress bar.	".flc-progress-bar"
`indicator`	[fluid:required] The element that represents the "progressor" as it grows over time.	".flc-progress-indicator"
`label`	[fluid:optional] The container for the progress bar's label.	".flc-progress-label"

`ariaElement`	[fluid:required, except in the case where another element on the page carries the data required to present progress information to the screen reader, such as the case where there is a total progress indicator, and a sub-total progress indicator]	".flc-progress-bar"

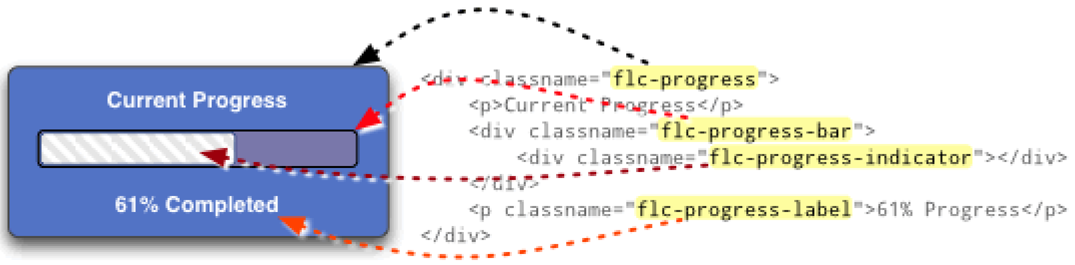
Any selectors not provided as an option will revert to the default. Implementers may choose to use the default class names in their markup, or customize the selectors, or a combination of these two approaches.

Example using default selectors:

```
selectors: {
    displayElement: ".flc-progress",
    progressBar: ".flc-progress-bar",
    indicator: ".flc-progress-indicator",
    label: ".flc-progress-label",
    ariaElement: ".flc-progress-bar"
}
```



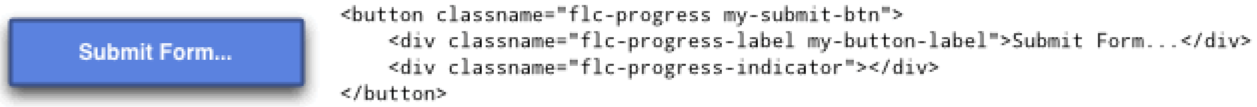
```
<div classname="flc-progress">
    <p>Current Progress</p>
    <div classname="flc-progress-bar">
        <div classname="flc-progress-indicator"></div>
    </div>
    <p classname="flc-progress-label">61% Progress</p>
</div>
```



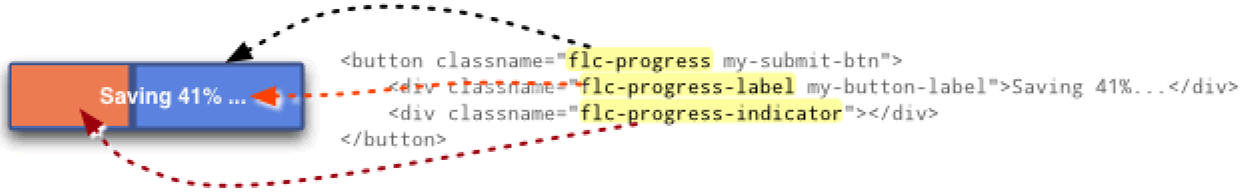
```
<div classname="flc-progress">
    <p>Current Progress</p>
    <div classname="flc-progress-bar">
        <div classname="flc-progress-indicator"></div>
    </div>
    <p classname="flc-progress-label">61% Progress</p>
</div>
```

Example mixing up the selectors a bit for a different effect:

```
selectors: {
    displayElement: ".flc-progress-indicator",
    progressBar: ".flc-progress",
    indicator: ".flc-progress-indicator",
    label: ".flc-progress-label",
    ariaElement: ".flc-progress"
},
```



```
<button classname="flc-progress my-submit-btn">
    <div classname="flc-progress-label my-button-label">Submit Form...</div>
    <div classname="flc-progress-indicator"></div>
</button>
```



```
<button classname="flc-progress my-submit-btn">
    <div classname="flc-progress-label my-button-label">Saving 41%...</div>
    <div classname="flc-progress-indicator"></div>
</button>
```

Dependencies

Progress dependencies can be met by including the minified `InfusionAll.js` file in the header of the HTML file.

```
<script type="text/javascript" src="InfusionAll.js"></script>
```

Alternatively, the individual file requirements are:

```
<script type="text/javascript" src="lib/jquery/core/js/jquery.js"></script>
<script type="text/javascript" src="lib/jquery/ui/js/ui.core.js"></script>
<script type="text/javascript" src="lib/jquery/plugins/bgiframe/js/jquery.bgiframe.js"></script>  <!-- New in v1.3 -->
<script type="text/javascript" src="framework/core/js/Fluid.js"></script>
<script type="text/javascript" src="components/progress/js/Progress.js"></script>
```