DOM Elements

React implements a browser-independent DOM system for performance and cross-browser compatibility. We took the opportunity to clean up a few rough edges in browser DOM implementations.

In React, all DOM properties and attributes (including event handlers) should be camelCased. For example, the HTML attribute `tabindex` corresponds to the attribute `tabIndex` in React. The exception is `aria-*` and `data-*` attributes, which should be lowercased. For example, you can keep `aria-label` as `aria-label`.

Differences In Attributes

There are a number of attributes that work differently between React and HTML:

checked

The `checked` attribute is supported by `<input>` components of type `checkbox` or `radio`. You can use it to set whether the component is checked. This is useful for building controlled components. `defaultChecked` is the uncontrolled equivalent, which sets whether the component is checked when it is first mounted.

className

To specify a CSS class, use the `className` attribute. This applies to all regular DOM and SVG elements like `<div>`, `<a>`, and others.

If you use React with Web Components (which is uncommon), use the `class` attribute instead.

dangerouslySetInnerHTML

`dangerouslySetInnerHTML` is React's replacement for using `innerHTML` in the browser DOM. In general, setting HTML from code is risky because it's easy to inadvertently expose your users to a cross-site scripting (XSS) attack. So, you can set HTML directly from React, but you have to type out `dangerouslySetInnerHTML` and pass an object with a `__html` key, to remind yourself that it's dangerous. For example:

```
function createMarkup() {
return {__html: 'First &middot; Second'};
}

function MyComponent() {
return <div dangerouslySetInnerHTML={createMarkup()} />;
}
```

htmlFor

Since `for` is a reserved word in JavaScript, React elements use `htmlFor` instead.

onChange

The `onChange` event behaves as you would expect it to: whenever a form field is changed, this event is fired. We intentionally do not use the existing browser behavior because `onChange` is a misnomer for its behavior and React relies on this event to handle user input in real time.

selected

The `selected` attribute is supported by `<option>` components. You can use it to set whether the component is selected. This is useful for building controlled components.

style

> Note
>
> Some examples in the documentation use `style` for convenience, but **using the `style` attribute as the primary means of styling elements is generally not recommended.** In most cases, `className` should be used to reference classes defined in an external CSS stylesheet. `style` is most often used in React applications to add dynamically-computed styles at render time. See also FAQ: Styling and CSS.

The `style` attribute accepts a JavaScript object with camelCased properties rather than a CSS string. This is consistent with the DOM `style` JavaScript property, is more efficient, and prevents XSS security holes. For example:

```
const divStyle = {
color: 'blue',
backgroundImage: 'url(' + imgUrl + ')',
};

function HelloWorldComponent() {
return <div style={divStyle}>Hello World!</div>;
}
```

Note that styles are not autoprefixed. To support older browsers, you need to supply corresponding style properties:

```
const divStyle = {
WebkitTransition: 'all', // note the capital 'W' here
msTransition: 'all' // 'ms' is the only lowercase vendor prefix
};

function ComponentWithTransition() {
return <div style={divStyle}>This should work cross-browser</div>;
}
```

Style keys are camelCased in order to be consistent with accessing the properties on DOM nodes from JS (e.g. `node.style.backgroundImage`). Vendor prefixes other than `ms` should begin with a capital letter. This is why `WebkitTransition` has an uppercase "W".

React will automatically append a "px" suffix to certain numeric inline style properties. If you want to use units other than "px", specify the value as a string with the desired unit. For example:

```
// Result style: '10px'
<div style={{ height: 10 }}>
Hello World!
</div>

// Result style: '10%'
<div style={{ height: '10%' }}>
Hello World!
</div>
```

Not all style properties are converted to pixel strings though. Certain ones remain unitless (eg `zoom`, `order`, `flex`). A complete list of unitless properties can be seen here.

suppressContentEditableWarning

Normally, there is a warning when an element with children is also marked as `contentEditable`, because it won't work. This attribute suppresses that warning. Don't use this unless you are building a library like Draft.js that manages `contentEditable` manually.

suppressHydrationWarning

If you use server-side React rendering, normally there is a warning when the server and the client render different content. However, in some rare cases, it is very hard or impossible to guarantee an exact match. For example, timestamps are expected to differ on the server and on the client.

If you set `suppressHydrationWarning` to `true`, React will not warn you about mismatches in the attributes and the content of that element. It only works one level deep, and is intended to be used as an escape hatch. Don't overuse it. You can read more about hydration in the `ReactDOM.hydrate()` documentation.

value

The `value` attribute is supported by `<input>` and `<textarea>` components. You can use it to set the value of the component. This is useful for building controlled components. `defaultValue` is the uncontrolled equivalent, which sets the value of the component when it is first mounted.

All Supported HTML Attributes

As of React 16, any standard or custom DOM attributes are fully supported.

React has always provided a JavaScript-centric API to the DOM. Since React components often take both custom and DOM-related props, React uses the `camelCase` convention just like the DOM APIs:

```
<div tabIndex="-1" />      // Just like node.tabIndex DOM API
<div className="Button" /> // Just like node.className DOM API
<input readOnly={true} />  // Just like node.readOnly DOM API
```

These props work similarly to the corresponding HTML attributes, with the exception of the special cases documented above.

Some of the DOM attributes supported by React include:

```
accept acceptCharset accessKey action allowFullScreen alt async autoComplete
autoFocus autoPlay capture cellPadding cellSpacing challenge charSet checked
cite classID className colSpan cols content contentEditable contextMenu controls
controlsList coords crossOrigin data dateTime default defer dir disabled
download draggable encType form formAction formEncType formMethod formNoValidate
formTarget frameBorder headers height hidden high href hrefLang htmlFor
httpEquiv icon id inputMode integrity is keyParams keyType kind label lang list
loop low manifest marginHeight marginWidth max maxLength media mediaGroup method
min minLength multiple muted name noValidate nonce open optimum pattern
placeholder poster preload profile radioGroup readOnly rel required reversed
role rowSpan rows sandbox scope scoped scrolling seamless selected shape size
sizes span spellCheck src srcDoc srcLang srcSet start step style summary
tabIndex target title type useMap value width wmode wrap
```

Similarly, all SVG attributes are fully supported:

```
accentHeight accumulate additive alignmentBaseline allowReorder alphabetic
amplitude arabicForm ascent attributeName attributeType autoReverse azimuth
baseFrequency baseProfile baselineShift bbox begin bias by calcMode capHeight
clip clipPath clipPathUnits clipRule colorInterpolation
colorInterpolationFilters colorProfile colorRendering contentScriptType
contentStyleType cursor cx cy d decelerate descent diffuseConstant direction
display divisor dominantBaseline dur dx dy edgeMode elevation enableBackground
end exponent externalResourcesRequired fill fillOpacity fillRule filter
filterRes filterUnits floodColor floodOpacity focusable fontFamily fontSize
fontSizeAdjust fontStretch fontStyle fontVariant fontWeight format from fx fy
g1 g2 glyphName glyphOrientationHorizontal glyphOrientationVertical glyphRef
gradientTransform gradientUnits hanging horizAdvX horizOriginX ideographic
imageRendering in in2 intercept k k1 k2 k3 k4 kernelMatrix kernelUnitLength
kerning keyPoints keySplines keyTimes lengthAdjust letterSpacing lightingColor
limitingConeAngle local markerEnd markerHeight markerMid markerStart
markerUnits markerWidth mask maskContentUnits maskUnits mathematical mode
numOctaves offset opacity operator order orient orientation origin overflow
```

```
overlinePosition overlineThickness paintOrder panose1 pathLength
patternContentUnits patternTransform patternUnits pointerEvents points
pointsAtX pointsAtY pointsAtZ preserveAlpha preserveAspectRatio primitiveUnits
r radius refX refY renderingIntent repeatCount repeatDur requiredExtensions
requiredFeatures restart result rotate rx ry scale seed shapeRendering slope
spacing specularConstant specularExponent speed spreadMethod startOffset
stdDeviation stemh stemv stitchTiles stopColor stopOpacity
strikethroughPosition strikethroughThickness string stroke strokeDasharray
strokeDashoffset strokeLinecap strokeLinejoin strokeMiterlimit strokeOpacity
strokeWidth surfaceScale systemLanguage tableValues targetX targetY textAnchor
textDecoration textLength textRendering to transform u1 u2 underlinePosition
underlineThickness unicode unicodeBidi unicodeRange unitsPerEm vAlphabetic
vHanging vIdeographic vMathematical values vectorEffect version vertAdvY
vertOriginX vertOriginY viewBox viewTarget visibility widths wordSpacing
writingMode x x1 x2 xChannelSelector xHeight xlinkActuate xlinkArcrole
xlinkHref xlinkRole xlinkShow xlinkTitle xlinkType xmlns xmlnsXlink xmlBase
xmlLang xmlSpace y y1 y2 yChannelSelector z zoomAndPan
```

You may also use custom attributes as long as they’re fully lowercase.

Edit this page