The `Element` property **`innerHTML`** gets or sets the HTML or XML markup contained within the element.

> **Note:** If a `<div>`, `<span>`, or `<noembed>` node has a child text node that includes the characters `(&)`, `(<)`, or `(>)`, `innerHTML` returns these characters as the HTML entities `"&amp;"`, `"&lt;"` and `"&gt;"` respectively. Use `Node.textContent` to get a raw copy of these text nodes' contents.

To insert the HTML into the document rather than replace the contents of an element, use the method `insertAdjacentHTML()`.

Syntax

```
const content = element.innerHTML;

element.innerHTML = htmlString;
```

Value

A `DOMString` containing the HTML serialization of the element's descendants. Setting the value of `innerHTML` removes all of the element's descendants and replaces them with nodes constructed by parsing the HTML given in the string *htmlString*.

Exceptions

`SyntaxError`
An attempt was made to set the value of `innerHTML` using a string which is not properly-formed HTML.

`NoModificationAllowedError`
An attempt was made to insert the HTML into a node whose parent is a `Document`.

Usage notes

The `innerHTML` property can be used to examine the current HTML source of the page, including any changes that have been made since the page was initially loaded.

Reading the HTML contents of an element

Reading `innerHTML` causes the user agent to serialize the HTML or XML fragment comprised of the element's descendants. The resulting string is returned.

```
let contents = myElement.innerHTML;
```

This lets you look at the HTML markup of the element's content nodes.

> **Note:** The returned HTML or XML fragment is generated based on the current contents of the element, so the markup and formatting of the returned fragment is likely not to match the original page markup.

Replacing the contents of an element

Setting the value of `innerHTML` lets you easily replace the existing contents of an element with new content.

For example, you can erase the entire contents of a document by clearing the contents of the document's `body` attribute:

```
document.body.innerHTML = "";
```

This example fetches the document's current HTML markup and replaces the `"<"` characters with the HTML entity `"<"`, thereby essentially converting the HTML into raw text. This is then wrapped in a `<pre>` element. Then the value of `innerHTML` is changed to this new string. As a result, the document contents are replaced with a display of the page's entire source code.

```
document.documentElement.innerHTML = "<pre>" +
         document.documentElement.innerHTML.replace(/</g,"&lt;") +
            "</pre>";
```

Operational details

What exactly happens when you set value of `innerHTML`? Doing so causes the user agent to follow these steps:

1. The specified value is parsed as HTML or XML (based on the document type), resulting in a `DocumentFragment` object representing the new set of DOM nodes for the new elements.
2. If the element whose contents are being replaced is a `<template>` element, then the `<template>` element's `content` attribute is replaced with the new `DocumentFragment` created in step 1.
3. For all other elements, the element's contents are replaced with the nodes in the new `DocumentFragment`.

Security considerations

It is not uncommon to see `innerHTML` used to insert text into a web page. There is potential for this to become an attack vector on a site, creating a potential security risk.

```
const name = "John";
// assuming 'el' is an HTML DOM element
el.innerHTML = name; // harmless in this case

// ...

name = "<script>alert('I am John in an annoying alert!')</script>";
el.innerHTML = name; // harmless in this case
```

Although this may look like a cross-site scripting attack, the result is harmless. HTML5 specifies that a `<script>` tag inserted with `innerHTML` should not execute.

However, there are ways to execute JavaScript without using `<script>` elements, so there is still a security risk whenever you use `innerHTML` to set strings over which you have no control. For example:

```
const name = "<img src='x' onerror='alert(1)'>";
el.innerHTML = name; // shows the alert
```

For that reason, it is recommended that you do not use `innerHTML` when inserting plain text; instead, use `Node.textContent`. This doesn't parse the passed content as HTML, but instead inserts it as raw text.

> **Warning:** If your project is one that will undergo any form of security review, using `innerHTML` most likely will result in your code being rejected. For example, if you

> use `innerHTML` in a browser extension and submit the extension to addons.mozilla.org, it will not pass the automated review process.

Example

This example uses `innerHTML` to create a mechanism for logging messages into a box on a web page.

JavaScript

```
function log(msg) {
  var logElem = document.querySelector(".log");

  var time = new Date();
  var timeStr = time.toLocaleTimeString();
  logElem.innerHTML += timeStr + ": " + msg + "<br/>";
}

log("Logging mouse events inside this container...");
```

The `log()` function creates the log output by getting the current time from a `Date` object using `toLocaleTimeString()`, and building a string with the timestamp and the message text. Then the message is appended to the box with the class `"log"`.

We add a second method that logs information about `MouseEvent` based events (such as `mousedown`, `click`, and `mouseenter`):

```
function logEvent(event) {
  var msg = "Event <strong>" + event.type + "</strong> at <em>" +
            event.clientX + ", " + event.clientY + "</em>";
  log(msg);
}
```

Then we use this as the event handler for a number of mouse events on the box that contains our log:

```
var boxElem = document.querySelector(".box");

boxElem.addEventListener("mousedown", logEvent);
boxElem.addEventListener("mouseup", logEvent);
boxElem.addEventListener("click", logEvent);
boxElem.addEventListener("mouseenter", logEvent);
boxElem.addEventListener("mouseleave", logEvent);
```

HTML

The HTML is quite simple for our example.

```
<div class="box">
  <div><strong>Log:</strong></div>
  <div class="log"></div>
</div>
```

The `<div>` with the class `"box"` is just a container for layout purposes, presenting the contents with a box around it. The `<div>` whose class is `"log"` is the container for the log text itself.

CSS

The following CSS styles our example content.

```
.box {
  width: 600px;
  height: 300px;
  border: 1px solid black;
  padding: 2px 4px;
  overflow-y: scroll;
  overflow-x: auto;
}

.log {
  margin-top: 8px;
  font-family: monospace;
}
```

Result

The resulting content looks like this. You can see output into the log by moving the mouse in and out of the box, clicking in it, and so forth.