DOMParser

`DOMParser` can be used to parse strings and streams of XML text. It can't be used to parse HTML "tag soup" (see Parsing HTML from chrome if you need that).

`DOMParser` is available to web pages, but it's not part of any standard and is not supported by some browsers (but see this workaround). Note, that if you want to parse XML received from the server, you can just use the `responseXML` property of XMLHttpRequest.

The documentation is in the IDL: `content/base/public/nsIDOMParser.idl`.

Table of contents

Creating a DOMParser

To create a `DOMParser` object from a web page or a chrome script running in a window, simply use `new DOMParser()`. When you create a `DOMParser` from a privileged script, you can pass parameters to the constructor, more on that below.

To create a `DOMParser` when the constructor is not available (e.g., from a JS XPCOM component, a JS module, or an xpcshell test), use:

```
view plain  print  ?
01. var parser = Components.classes["@mozilla.org/xmlextras/domparser;1"]
02.              .createInstance(Components.interfaces.nsIDOMParser);
03. // optionally, call parser.init(principal, documentURI, baseURI);
```

Parsing XML

Once you have created a parser object, parsing is easy:

```
view plain  print  ?
01. var parser = new DOMParser();
02. var doc = parser.parseFromString(aStr, "text/xml");
```

Error handling

Note that if the parsing process failed, `DOMParser` currently does not throw an exception, but instead returns an error document (see bug 45566):

```
view plain  print  ?
01. <parsererror xmlns="http://www.mozilla.org/newlayout/xml/parsererror.xml">
02. (error description)
03. <sourcetext>(a snippet of the source XML)</sourcetext>
04. </parsererror>
```

The parsing errors are also reported to the Error Console, with the document URI (see below) as the source of the error.

Principals, document and base URI

Note: This section covers changes introduced to `DOMParser` in Gecko 1.9.

To create a document, the parser needs to specify a principal (see Security check basics), a base URI (see document.baseURIObject), and a documentURI.

These values are automatically determined as defined below, but if you work with `DOMParser` from privileged code, you can override the defaults by providing arguments to the DOMParser constructor or calling `parser.init()`. Usually you don't need to do that. If you come across a situation when these matter, feel free to ask questions in mozilla.dev.tech.dom and update this documentation to mention these cases.

- When a `DOMParser` is instantiated by calling `new DOMParser()`, it inherits the calling code's principal (except that for chrome callers the principal is set to the null principal) and the `documentURI` and `baseURI` of the window the constructor came from.
- If the caller has UniversalXPConnect privileges, it can pass parameters to `new DOMParser()`. If fewer than three parameters are passed, the remaining parameters will default to `null`.
  - The first parameter is the principal to use; this overrides the default principal normally inherited.
  - The second parameter is the `documentURI` to use.
  - The third parameter is the `baseURI` to use.
- If you instantiate a `DOMParser` by calling `createInstance()`, and you don't call the `DOMParser`'s `init()` method, attempting to initiate a parsing operation will automatically call `init()` on the `DOMParser` with a null principal and `null` pointers for `documentURI` and `baseURI`.

Cases where these values matter:

- If you don't specify the document URI by calling init() after creating the parser via `createInstance()` the created documents will use a moz-nullprincipal:{<guid>} URI, which will show in the Error Console in parsing errors, in particular.
- Supposedly, if you want to create objects that some particular set of unprivileged code will be able to access (see discussion in bug 565480).

Examples

Within the context of a window:


```
view plain  print  ?
01. var parser = new DOMParser();
02. var doc = parser.parseFromString(aStr, "text/xml");
```

Outside of a window (e.g., a JS XPCOM component, a JS module, or an xpcshell test):

```
view plain  print  ?
01. var parser = Components.classes["@mozilla.org/xmlextras/domparser;1"]
02.              .createInstance(Components.interfaces.nsIDOMParser);
03. var doc = parser.parseFromString(aStr, "text/xml");
```

Using `Components.Constructor()`:

```
view plain  print  ?
01. const DOMParser = new Components.Constructor("@mozilla.org/xmlextras/domparser;1", "nsIDOMParser");
02. var parser = new DOMParser();
03. parser.init(principal, documentURI, baseURI);
04. var doc = parser.parseFromString(aStr, "text/xml");
```

See also

- Parsing and serializing XML
- XMLHttpRequest
- XMLSerializer
- Parsing HTML from chrome

Page last modified 22:13, 22 May 2010 by Nickolay