HTML5의 DOM으로 추가된 File API를 사용하여, 이제 웹 컨텐츠가 사용자에게 로컬 파일을 선택한 후 파일의 컨텐츠를 읽도록 요청할 수 있습니다. 이 선택은 HTML `<input>` 엘리먼트나 드래그 앤 드랍을 사용하는 것으로도 수행할 수 있습니다.

원하신다면 확장 기능이나 다른 브라우저 크롬 코드에서도 DOM File API를 사용하실 수 있습니다. 하지만, 조심해야할 몇 가지 부가적인 기능들이 있음을 유의하세요. 자세한 내용은 크롬 코드에서 DOM File API 사용하기 글을 확인하세요.

선택한 파일에 접근하기1

다음 HTML을 생각해봅시다.

```
<input type="file" id="input">
```

File API는 사용자에 의해 선택된 파일을 나타내는 객체인 `File`을 포함하는 `FileList`에 접근할 수 있게 해줍니다.

사용자가 하나의 파일만을 선택한 경우, 리스트의 첫 번째 파일만 고려하면 됩니다.

기존의 DOM 셀렉터를 사용하여 선택된 하나의 파일에 접근하기:

```
const selectedFile = document.getElementById('input').files[0];
```

change 이벤트에서 선택한 파일에 접근하기

`change` 이벤트를 통해 `FileList`에 접근할수도 있습니다(필수는 아닙니다).

```
<input type="file" id="input" onchange="handleFiles(this.files)"
>
```

사용자가 하나의 파일을 선택할 때, 사용자에 의해 선택된 파일을 나타내는 객체인 `File`을 포함하는 `FileList`와 함께 `handlerFiles()` 함수가 호출됩니다.

사용자가 여러 파일을 선택할 수 있도록 하길 원할 경우, 간단히 `input` 엘리먼트에서 `multiple` 속성을 사용하면됩니다.

```
```

```
<input type="file" id="input" multiple onchange="handleFiles
(this.files)">
```

이 경우, `handleFiles()` 함수로 전달된 파일 리스트는 사용자가 선택한 각 파일에 대해 하나의 `File` 객체를 갖습니다.

동적으로 change 리스너 추가하기

`change` 이벤트 리스너를 추가하려면 `EventTarget.addEventListener()`를 다음과 같이 사용해야합니다.

```
const inputElement = document.getElementById("input");
inputElement.addEventListener("change", handleFiles, false);
function handleFiles() { const fileList = this.files; /* ## ## #
### ### # #### */ }
```

이 경우에는, 파라미터를 전달한 이벤트 핸들러에의해 호출된 이전 예제에서와 달리, `handleFiles()` 함수 자체가 이벤트 핸들러임을 유의하세요.

선택한 파일에 대한 정보 얻기

DOM에 의해 제공된 `FileList` 객체는 사용자에 의해 선택된 모든 파일을 각각 `File` 객체로 지정하여 나열합니다. 파일 리스트의 `length` 속성의 값을 확인하여 사용자가 선택한 파일의 수를 결정할 수 있습니다.

```
const numFiles = files.length;
```

개별 `File` 객체는 리스트를 간단히 배열처럼 접근하여 얻을 수 있습니다.

```
for (let i = 0, numFiles = files.length; i < numFiles; i++) {
const file = files[i]; .. }
```

위 반복문은 파일 리스트의 모든 파일을 순회합니다.

파일에 대한 유용한 정보를 포함하는 `File` 객체는 세 가지 속성을 제공합니다.

name
읽기 전용 문자열인 파일의 이름입니다. 단순한 파일 이름이며, 경로에 대한 정보는 포함하지 않습니다.

size
읽기 전용 64비트 정수의 바이트 단위 파일의 크기입니다.

type
읽기 전용 문자열인 파일의 MIME 타입입니다. 결정할 수 없는 타입인 경우 "" 입니다.

예시: 파일 크기 보기

다음 예시는 size 프로퍼티를 사용하는 방법을 보여줍니다.

```
<!DOCTYPE html> <html> <head> <meta charset="UTF-8"> <title>File(s) size</title> <script> function updateSize() { let nBytes = 0, oFiles = document.getElementById("uploadInput").files, nFiles = oFiles.length; for (let nFileId = 0; nFileId < nFiles; nFileId++) { nBytes += oFiles[nFileId].size; } let sOutput = nBytes + " bytes"; // multiples approximation# ## ### ## for (let aMultiples = ["KiB", "MiB", "GiB", "TiB", "PiB", "EiB", "ZiB", "YiB"], nMultiple = 0, nApprox = nBytes / 1024; nApprox > 1; nApprox /= 1024, nMultiple++) { sOutput = nApprox.toFixed(3) + " " + aMultiples[nMultiple] + " (" + nBytes + " bytes)"; } // ### # ### # document.getElementById("fileNum").innerHTML = nFiles; document.getElementById("fileSize").innerHTML = sOutput; } </script> </head> <body onload="updateSize();"> <form name="uploadForm"> <p><input id="uploadInput" type="file" name="myFiles" onchange="updateSize();" multiple> selected files: <span id="fileNum">0</span>; total size: <span id="fileSize">0</span></p> <p><input type="submit" value="Send file"></p> </form> </body> </html>
```

---

click() 메소드를 사용하여 숨겨진 파일 input 엘리먼트 사용하기

Gecko 2.0 (Firefox 4 / Thunderbird 3.3 / SeaMonkey 2.1)의 시작과 함께, 확실히 세련되지 않은 파일 <input> 엘리먼트를 숨기고 파일 선택기를 열고 사용자에 의해 선택된 파일 또는 파일들을 보여주는 여러분만의 인터페이스를 제공할 수 있습니다. input 엘리먼트를 display:none 으로 스타일링하고 <input> 엘리먼트에 click() 메소드를 호출하는 것으로 이를 수행할 수 있습니다.

다음 HTML을 생각해봅시다.

```
<input type="file" id="fileElem" multiple accept="image/*" style
="display:none" onchange="handleFiles(this.files)"> <button id="
fileSelect">Select some files</button>
```

`click` 이벤트를 다루는 코드는 다음과 같을 것입니다.

```
const fileSelect = document.getElementById("fileSelect"),
fileElem = document.getElementById("fileElem"); fileSelect.
addEventListener("click", function (e) { if (fileElem) {
fileElem.click(); } }, false);
```

여러분이 원하는 파일 선택기를 열기위한 새로운 버튼을 스타일링할 수 있습니다.

---

label 엘리먼트를 사용하여 숨겨진 파일 input 엘리먼트 실행하기

JavaScript(click() 메소드)를 사용하지 않고 파일 선택기를 열도록 허용하기 위해 `<label>` 엘리먼트가 사용될 수 있습니다. 이 경우에는 input 엘리먼트가 반드시 `display: none`(또는 `visibility: hidden`)을 사용하여 숨긴상태가 아니어야하며, 그렇지 않을 경우 라벨은 키보드로 접근이 불가하다는 것을 유의하세요. 대신 외관상으로 숨기기 테크닉을 사용하세요.

다음 HTML과

```
<input type="file" id="fileElem" multiple accept="image/*" class
="visually-hidden"> <label for="fileElem">Select some files</
label>
```

CSS를 생각해봅시다.

```
.visually-hidden { position: absolute !important; height: 1px;
width: 1px; overflow: hidden; clip: rect(1px, 1px, 1px, 1px); }
input.visually-hidden:focus + label { outline: thin dotted; }
```

`fileElem.click()`을 호출하기위해 JavaScript 코드를 추가할 필요가 없습니다. 또한 이 경우에는 여러분이 원하는대로 label 엘리먼트를 스타일링 할 수 있습니다. 여러

분은 숨겨진 input 필드의 포커싱 상태를 시각적인 신호(위에서 보여진 outline이나, background-color 또는 box-shadow)로 label에 제공해야합니다. (이 글의 작성 시점에서, Firefox는 `<input type="file">` 엘리먼트에 대한 시각적 신호를 보여주지 않습니다.)

드래그 앤 드랍을 사용하여 파일 선택하기

사용자가 파일을 웹 어플리케이션으로 드래그 앤 드랍하도록 할 수도 있습니다.

첫 단계는 드랍 영역을 설정하는 것입니다. 드랍을 허용할 컨텐츠의 정확한 영역은 어플리케이션의 디자인에따라 아주 달라질 수 있지만, 드랍 이벤트를 받는 엘리먼트를 만드는 것은 간단합니다.

```
let dropbox; dropbox = document.getElementById("dropbox");
dropbox.addEventListener("dragenter", dragenter, false); dropbox
.addEventListener("dragover", dragover, false); dropbox.
addEventListener("drop", drop, false);
```

이 예시에서는, `dropbox`라는 ID를 갖는 엘리먼트를 드랍 영역으로 변경합니다. `dragenter`, `dragover`, `drop` 이벤트를위한 리스너를 추가하는 것으로 이를 수행할 수 있습니다.

우리의 경우에는, `dragenter`와 `dragover` 이벤트로 무언가를 진짜 할 필요는 없으므로, 두 함수는 모두 단순합니다. 두 함수는 단지 이벤트의 전파를 중단하고 기본 동작이 발생하는 것을 방지합니다.

```
function dragenter(e) { e.stopPropagation(); e.preventDefault();
} function dragover(e) { e.stopPropagation(); e.preventDefault()
; }
```

진짜 마법은 `drop()` 함수에서 발생합니다.

```
function drop(e) { e.stopPropagation(); e.preventDefault();
const dt = e.dataTransfer; const files = dt.files; handleFiles(
files); }
```

여기에서, 우리는 이벤트로부터 `dataTransfer` 필드를 추출하고, 그로부터 파일 리스트를 가져온 후, `handleFiles()`로 전달합니다. 이 지점부터, 파일을 다루는 것은 사용자가 `input` 엘리먼트를 사용했든 드래그 앤 드랍을 사용했든 동일합니다.

예시: 사용자가 선택한 이미지의 섬네일 보여주기

여러분이 차세대 사진 공유 웹사이트를 개발중이며 HTML5를 사용하여 사진이 실제로 업로드되기 전에 이미지의 섬네일 미리보기를 표시하길 원한다고 가정해봅시다. 여러분은 앞서 설명한대로 input 엘리먼트나 드랍 영역을 설정하고 아래와 같은 `handleFiles()` 함수를 호출하면됩니다.

```
function handleFiles(files) { for (let i = 0; i < files.length; i++) { const file = files[i]; if (!file.type.startsWith('image/')){ continue } const img = document.createElement("img"); img.classList.add("obj"); img.file = file; preview.appendChild(img); // "preview"# ### ### div #### ##. const reader = new FileReader(); reader.onload = (function(aImg) { return function(e) { aImg.src = e.target.result; }; })(img); reader.readAsDataURL(file); } }
```

H여기에서 사용자가 선택한 파일을 다루는 반복문은 각 파일의 `type` 속성을 보고 MIME 타입이 "`image/`" 문자열로 시작하는지를 확인합니다. 이미지인 각 파일에 대해서는, 새로운 `img` 엘리먼트를 생성합니다. CSS를 사용하여 보기 좋은 테두리나 그림자를 설정할 수 있고 이미지의 크기를 지정할 수 있으므로, 스타일링에 대해서는 여기에서 다룰 필요는 없습니다.

각 이미지는 각각에 추가된 CSS 클래스 `obj`를 가져, DOM 트리에서의 탐색을 더 쉽게만듭니다. 각 이미지에 이미지에 대한 `File`을 지정하는 `file` 속성도 추가합니다 (이는 나중에 실제로 업로드를 위한 이미지를 fetch 할 수 있게해줍니다). `Node.appendChild()`를 사용하여 다큐먼트의 미리보기 영역에 새로운 섬네일을 추가합니다.

다음으로, `FileReader`를 설정하여 이미지 로딩과 이를 `img` 엘리먼트에 추가하는 것을 비동기적으로 처리합니다. 새로운 `FileReader` 객체를 생성한 후에, `onload` 함수를 설정하고 `readAsDataURL()`을 호출하여 백그라운드에서 읽기 작업을 시작합니다. 이미지 파일의 전체 컨텐츠가 로드되었을 때, onload 콜백으로 전달되는 `data:` URL로 변환됩니다. 이 루틴을 구현하면 img 엘리먼트의 src 속성을 로드된 이미지로 설정하여 사용자 화면의 섬네일에 이미지가 나타나납니다.

객체 URL 사용하기

Gecko 2.0 (Firefox 4 / Thunderbird 3.3 / SeaMonkey 2.1)은 DOM `window.URL.createObjectURL()` 및 `window.URL.revokeObjectURL()` 메소

드에 대한 지원을 소개했습니다. 이 메소드들은 사용자의 컴퓨터에 있는 로컬 파일을 포함해, DOM `File` 객체를 사용해 참조된 데이터에 대한 참조로 사용할 수 있는 간단한 URL 문자열을 생성할 수 있게 해줍니다.

HTML에 URL로 참조하길 원하는 `File` 객체가 있다면, 다음과 같이 객체 URL을 생성할 수 있습니다.

```
const objectURL = window.URL.createObjectURL(fileObj);
```

객체 URL은 `File` 객체를 식별하는 문자열입니다. `window.URL.createObjectURL()`을 호출할때마다, 여러분이 이미 해당 파일에 대한 객체 URL을 생성했더라도 고유한 객체 URL이 생성됩니다. 각각은 반드시 해제되어야 합니다. 객체 URL은 다큐먼트가 unload될 때 자동으로 해제되지만, 여러분의 페이지가 동적으로 이를 사용할 경우 `window.URL.revokeObjectURL()`을 호출하여 명시적으로 해제해야합니다.

```
window.URL.revokeObjectURL(objectURL);
```

예시: 객체 URL을 사용하여 이미지 표시하기

다음 예시는 객체 URL을 사용하여 이미지 섬네일을 표시합니다. 부가적으로, 파일의 이름과 크기를 포함한 다른 정보도 표시합니다.

인터페이스를 나타내는 HTML은 다음과 같습니다.

```
<input type="file" id="fileElem" multiple accept="image/*" style
="display:none" onchange="handleFiles(this.files)"> <a href="#"
id="fileSelect">Select some files</a> <div id="fileList"> <p>No
files selected!</p> </div>
```

위 코드는 파일 선택기를 불러오는 링크와 우리의 파일 `<input>` 엘리먼트를 설정합니다(파일 input을 숨겨 덜 매력적인 사용자 인터페이스가 표시되는 것을 방지하였으므로). 이는 파일 선택기를 불러오는 메소드와 마찬가지로, Using hidden file input elements using the click() method 섹션에서 설명합니다.

`handleFiles()` 메소드는 다음과 같습니다.

```
window.URL = window.URL || window.webkitURL;

const fileSelect = document.getElementById("fileSelect"),
```

```
    fileElem = document.getElementById("fileElem"),
    fileList = document.getElementById("fileList");

fileSelect.addEventListener("click", function (e) {
  if (fileElem) {
    fileElem.click();
  }
  e.preventDefault(); // "#" 해시로 이동을 방지
}, false);

function handleFiles(files) {
  if (!files.length) {
    fileList.innerHTML = "<p>No files selected!</p>";
  } else {
    fileList.innerHTML = "";
    const list = document.createElement("ul");
    fileList.appendChild(list);
    for (let i = 0; i < files.length; i++) {
      const li = document.createElement("li");
      list.appendChild(li);

      const img = document.createElement("img");
      img.src = window.URL.createObjectURL(files[i]);
      img.height = 60;
      img.onload = function() {
        window.URL.revokeObjectURL(this.src);
      }
      li.appendChild(img);
      const info = document.createElement("span");
      info.innerHTML = files[i].name + ": " + files[i].size + " bytes";
      li.appendChild(info);
    }
  }
}
```

`fileList` ID로 `<div>`의 URL을 페칭하는 것으로 시작합니다. 이는 섬네일을 포함하여 파일 리스트로 삽입하는 블록입니다.

`handleFiles()`로 전달된 `FileList` 객체가 `null`인 경우, 블록의 inner HTML을 간단하게 "No files selected!"를 표시하도록 설정합니다. `null`이 아닌 경우, 다음과 같이 파일 리스트를 구축합니다.

새로운 순서가 없는 리스트(`<ul>`) 엘리먼트가 생성됩니다.

새로운 리스트 엘리먼트가 `Node.appendChild()` 메소드 호출에 의해 `<div>` 블록으로 삽입됩니다.

`files`에 의해 나타나는 `FileList` 내의 각 `File`에 대해 :

1. 새로운 리스트 항목(`<li>`) 엘리먼트를 생성하고 리스트로 삽입합니다.
2. 새로운 이미지(`<img>`) 엘리먼트를 생성합니다.
3. `window.URL.createObjectURL()`을 사용하여 이미지의 소스를 파일을 나타내는 새로운 객체 URL로 설정해 blob URL을 생성합니다.
4. 이미지의 height를 60 픽셀로 설정합니다.
5. 이미지가 로드된 이후에 더 이상 필요하지 않게되므로 객체 URL을 해제하기 위한 이미지의 로드 이벤트 핸들러를 설정합니다. `window.URL.revokeObjectURL()` 메소드를 호출하고 `img.src`로 지정한 객체 URL 문자열을 전달하면됩니다.
6. 리스트로 새로운 리스트 항목을 추가합니다.

다음은 위 코드의 라이브 데모입니다.

Example: Uploading a user-selected file

Another thing you might want to do is let the user upload the selected file or files (such as the images selected using the previous example) to a server. This can be done asynchronously very easily.

Creating the upload tasks

Continuing with the code that built the thumbnails in the previous example, recall that every thumbnail image is in the CSS class `obj` with the corresponding `File` attached in a `file` attribute. This allows us to select all of the images the user has chosen for uploading using `Document.querySelectorAll()`, like this:

```
function sendFiles() { const imgs = document.querySelectorAll(".
obj"); for (let i = 0; i < imgs.length; i++) { new FileUpload(
imgs[i], imgs[i].file); } }
```

Line 2 fetches a `NodeList`, called `imgs`, of all the elements in the document with the CSS class `obj`. In our case, these will be all of the image thumbnails. Once we have that list, it's trivial to go through it and create a new `FileUpload` instance for each. Each of these handles uploading the corresponding file.

Handling the upload process for a file

The `FileUpload` function accepts two inputs: an image element and a file from which to read the image data.

```
function FileUpload(img, file) { const reader = new FileReader()
; this.ctrl = createThrobber(img); const xhr = new
XMLHttpRequest(); this.xhr = xhr; const self = this; this.xhr.
upload.addEventListener("progress", function(e) { if (e.
lengthComputable) { const percentage = Math.round((e.loaded *
100) / e.total); self.ctrl.update(percentage); } }, false); xhr.
upload.addEventListener("load", function(e){ self.ctrl.update(
100); const canvas = self.ctrl.ctx.canvas; canvas.parentNode.
removeChild(canvas); }, false); xhr.open("POST", "http://demos.
hacks.mozilla.org/paul/demos/resources/webservices/devnull.php")
; xhr.overrideMimeType('text/plain; charset=x-user-defined-
binary'); reader.onload = function(evt) { xhr.send(evt.target.
result); }; reader.readAsBinaryString(file); }
```

The `FileUpload()` function shown above creates a throbber, which is used to display progress information, and then creates an `XMLHttpRequest` to handle uploading the data.

Before actually transferring the data, several preparatory steps are taken:

1. The `XMLHttpRequest`'s upload `progress` listener is set to update the throbber with new percentage information so that as the upload progresses the throbber will be updated based on the latest information.
2. The `XMLHttpRequest`'s upload `load` event handler is set to update the throbber progress information to 100% to ensure the progress indicator actually reaches 100% (in case of granularity quirks during the process). It then removes the throbber since it's no longer needed. This causes the throbber to disappear once the upload is complete.
3. The request to upload the image file is opened by calling `XMLHttpRequest`'s `open()` method to start generating a POST request.
4. The MIME type for the upload is set by calling the `XMLHttpRequest` function `overrideMimeType()`. In this case, we're using a generic MIME type; you may or may not need to set the MIME type at all depending on your use case.
5. The `FileReader` object is used to convert the file to a binary string.
6. Finally, when the content is loaded the `XMLHttpRequest` function `send()` is called to upload the file's content.

> **Note:** The non-standard `sendAsBinary` method which was previously used in the example above is considered deprecated as of Gecko 31 (Firefox 31 / Thunderbird 31 / SeaMonkey 2.28); use the standard `send(Blob data)` method instead.

Asynchronously handling the file upload process

This example, which uses PHP on the server side and JavaScript on the client side, demonstrates asynchronous uploading of a file.

```
<?php if (isset($_FILES['myFile'])) { // Example:
move_uploaded_file($_FILES['myFile']['tmp_name'], "uploads/" .
$_FILES['myFile']['name']); exit; } ?><!DOCTYPE html> <html> <
head> <title>dnd binary upload</title> <meta http-equiv=
"Content-Type" content="text/html; charset=UTF-8"> <script type=
"application/javascript"> function sendFile(file) { const uri =
"/index.php"; const xhr = new XMLHttpRequest(); const fd = new
FormData(); xhr.open("POST", uri, true); xhr.onreadystatechange
= function() { if (xhr.readyState == 4 && xhr.status == 200) {
alert(xhr.responseText); // handle response. } }; fd.append(
'myFile', file); // Initiate a multipart/form-data upload xhr.
send(fd); } window.onload = function() { const dropzone =
document.getElementById("dropzone"); dropzone.ondragover =
dropzone.ondragenter = function(event) { event.stopPropagation()
; event.preventDefault(); } dropzone.ondrop = function(event) {
event.stopPropagation(); event.preventDefault(); const
filesArray = event.dataTransfer.files; for (let i=0; i<
filesArray.length; i++) { sendFile(filesArray[i]); } } } </
script> </head> <body> <div> <div id="dropzone" style="margin:
30px; width:500px; height:300px; border:1px dotted grey;">Drag &
drop your file here...</div> </div> </body> </html>
```

Example: Using object URLs to display PDF

Object URLs can be used for other things than just images! They can be used to display embedded PDF files or any other resources that can be displayed by the browser.

In Firefox, to have the PDF appear embedded in the iframe (rather than proposed as a downloaded file), the preference `pdfjs.disabled` must be set to `false` .

```
<iframe id="viewer">
```

And here is the change of the `src` attribute:

```
const obj_url = window.URL.createObjectURL(blob); const iframe =
document.getElementById('viewer'); iframe.setAttribute('src',
obj_url); window.URL.revokeObjectURL(obj_url);
```

Example: Using object URLs with other file types

You can manipulate files of other formats the same way. Here is how to preview uploaded video:

```
const video = document.getElementById('video'); const obj_url =
window.URL.createObjectURL(blob); video.src = obj_url; video.
play() window.URL.revokeObjectURL(obj_url);
```

Specifications

- File upload state (HTML 5 working draft)
- File API

See also

- `File`
- `FileList`
- `FileReader`
- Using XMLHttpRequest
- Using the DOM File API in chrome code
- `XMLHttpRequest`