HTML5 File API

HTML5 File API를 사용하면 로컬파일 정보 및 내용을 얻고 JavaScript로 처리할 수 있습니다.
앞으로 HTML5 의 보급과 클라우드 서비스가 진행되는 것에 따라 중요한 기능이 되는 것은 사실입니다.
다음 예제는 드래그 앤 드롭 기능이 적용된 파일 API 예제입니다. 단, 현재 Chrome 으로 로컬 작업시, 보안상 드러그앤드롭에 반응하지 않는 경우가 있습니다.

예제 소스코드

```
<!DOCTYPE html>
<html>
<head>
<meta charset="utf-8">
<title>File API 예제</title>
<style type="text/css">
label {
width : 200px;
}
</style>
<script>
var $id = function(id) { return document.getElementById(id); }
var $class = function(c, n) { n=n||0; return document.getElementsByClassName(c)[n]; }
var $classes = function(c) { return document.getElementsByClassName(c); }

window.onload = function()
{
// read 버튼 클릭시 실행하는 함수 등록
$id("read-button").addEventListener("change", onChangeFile, false);
// 드롭 이벤트 등록
$id("ta").addEventListener("dragover", onCancel, false);
$id("ta").addEventListener("dragenter", onCancel, false);
$id("ta").addEventListener("drop", onDropFile, false);
};

// 파일 변경시 이벤트
var onChangeFile = function(e)
{
```

```
// File 객체
var file = e.target.files[0];
// 파일 읽기
readFile(file);
};

// 파일 드롭시 이벤트
var onDropFile = function(e)
{
e.preventDefault();
// File 객체
var file = e.dataTransfer.files[0];
// 파일 읽기
readFile(file);
};

// 취소처리
var onCancel = function(e)
{
if(e.preventDefault) { e.preventDefault(); }
return false;
};

// 파일 읽기
var readFile = function(file)
{
var func_name = $id("func-name").value; // 함수명 얻기
var encode_type = $id("encode-type").value;// encode 타입 얻디
if (encode_type=="default") encode_type = undefined;

// 파일 정보 표시
$id("file-name").innerHTML = file.name; // 파일명
$id("file-size").innerHTML = file.size; // 파일크기
$id("file-type").innerHTML = file.type; // 파일타입

// 내용 읽기
var reader = new FileReader(); // FiileReader 생성
// 로드함수 등록
```

```
reader.onload = function(e) {
// 읽어든인 파일 내용을 텍스트영역에 설정
$id("ta").value = e.target.result;
};

// 텍스트로서 파일 읽음
// reader.readAsText(file);
// 실렉트 폼으로 선택하고 있는 함수명으로 파일읽기
reader[func_name](file, encode_type);
};

</script>
</head>
<body>
<h1>File API 예제</h1>

<p>Name : <span id="file-name">????</span></p>
<p>Size : <span id="file-size">????</span></p>
<p>Type : <span id="file-type">????</span></p>
<p>Value : </p>
<textarea id="ta" cols=64 rows=16></textarea>
<br />

<hr />

<label>
Func Name :
<select id="func-name">
<option value="readAsText">readAsText</option>
<option value="readAsDataURL">readAsDataURL</option>
<option value="readAsArrayBuffer">readAsArrayBuffer</option>
<option value="readAsBinaryString">readAsBinaryString</option>
</select>
</label>
<br />

<label>
Encode Type :
```

```
<select id="encode-type">
<option value="default">default</option>
<option value="utf-8">utf-8</option>
</select>
</label>
<br />

<input id="read-button" type="file" />
</body>
</html>
```

FileReader 객체에 대해

FileReader 객체는 파일을 읽어들이기 위한 객체입니다.
파일을 읽기 위한 함수는 총 4 개입니다.

- readAsText(blob, encoding)
- readAsDataUL(blob)
- readAsBinary(blob)
- readAsArrayBuffer(blob)

주로 많이 사용하는 것은 readAsText 와 readAsDataURL 이라고 봅니다.

매끄러운 파일 업로드 환경

파일 API를 사용하면 브라우저 또는 앱에서 파일을 읽고 조작할 수 있습니다.
단, 사용자의 승인이 필요합니다.
또한 파일 API는 플러그 인 없이도 매끄러운 파일 업로드 환경을 제공합니다.
다음 W3C 파일 API 예제에서 각 코드 블록은 진행, 오류 및 성공 상태를 처리합니다.

JavaScript :

```
function startRead() {
  // Obtain input element through DOM.

  var file = document.getElementById('file').files[0];
  if(file) {
    getAsText(file);
  }
}

function getAsText(readFile) {

  var reader = new FileReader();

  // Read file into memory as UTF-16
  reader.readAsText(readFile, "UTF-16");

  // Handle progress, success, and errors
  reader.onprogress = updateProgress;
  reader.onload = loaded;
  reader.onerror = errorHandler;
}

function updateProgress(evt) {
  if (evt.lengthComputable) {
    // evt.loaded and evt.total are ProgressEvent properties.
    var loaded = (evt.loaded / evt.total);

    if (loaded < 1) {
      // Increase the progress bar length.
      // style.width = (loaded * 200) + "px";
    }
```

```
  }
}

function loaded(evt) {
  // Obtain the read file data.
  var fileString = evt.target.result;

// Handle UTF-16 file dump
  if(utils.regexp.isChinese(fileString)) {
    //Chinese Characters + name validation.
  }
  else {
    // Run other charset test.
  }
  // xhr.send(fileString)
}

function errorHandler(evt) {
  if(evt.target.error.name == "NotReadableError") {
    // The file could not be read.
  }
}
```

Blob 생성자

Blob 생성자를 사용하면 웹 개발자가 클라이언트에서 직접 Blob(파일과 동등)를 생성하거나 조작할 수 있습니다.
생성자는 현재 작업 초안 단계에 있는 W3C 의 파일 API(영문) 사양에 정의되어 있습니다.
이전 버전의 파일 API 에서는 파일을 읽을 수만 있고 클라이언트에서 직접 편집할 수 없었습니다. 파일의 표현을 변경할 수는 있었지만 이러한 편집 내용을 저장하려면 파일 및 편집 내용을 서버에 업로드하고 서버에서 모든 작업을 처리한 다음 변경된 파일을 다운로드해야 했습니다.

Blob 생성자를 사용하면 배열에서 **Blob** 를 쉽게 만들 수 있습니다.
배열에는 **Blob** 개체, 텍스트 문자열 또는 배열 버퍼가 포함될 수 있으며, Blob 를 만들 때 배열을 매개 변수로 Blob 에 전달합니다.
선택적 사전 개체는 두 번째 매개 변수로 추가될 수 있으며, **type** 및 **endings** 라는 두 멤버를 포함할 수 있습니다.

다음 예제(Internet Explorer 10 이상)는 이 프로세스를 간단하게 보여 줍니다.
HTML :

```
<!DOCTYPE html>
<html>

<head>
  <meta content="text/html; charset=utf-8" http-equiv="Content-Type">
  <title>Saving Files Locally</title>
</head>

<body>
  <h1>Saving Files Locally</h1>
  <script>
    //check for Blob availability
    if (typeof Blob !== "undefined") {
      demoBlobs();
    } else {
      //your fallback code here
    }

    function demoBlobs(){
      //create a Blob with an array and the optional dictionary object.
      var blob1 = new Blob(
        ["There are 10 kinds of people ", "in the world.₩r₩n"], //array
        { type: "text/plain", endings: "native" } //dictionary object
      );

      //create a second blob that uses the first blob in its array
      var blob2 = new Blob([blob1, "Those who understand binary and those who don't."]);

      // Save the blob1 content to a file, giving just a "Save" option
      window.navigator.msSaveBlob(blob1, 'msSaveBlob_testFile.txt');
      alert('File saved using msSaveBlob() - note the single "Save" button below.');

      // Save the blob2 content to a file, giving both "Save" *and* "Open" options
      window.navigator.msSaveOrOpenBlob(blob2, 'msSaveBlobOrOpenBlob_testFile.txt');
      alert('File saved using msSaveOrOpenBlob() - note the two "Open" and "Save" buttons
below.');
    }
  </script>
```

</body>

</html>

msSaveBlob() 및 **msSaveOrOpenBlob()** 메서드를 통해 사용자는 웹에서 다운로드한 파일처럼 **Blob** 를 사용자 컴퓨터에 저장할 수 있습니다.
파일은 기본적으로 Downloads 폴더에 저장됩니다.
msSaveBlob()와 **msSaveOrOpenBlob()** 간의 차이점은 **msSaveBlob()** 메서드의 경우 **저장** 단추만 사용자에게 제공한다는 것입니다.
msSaveOrOpenBlob() 메서드는 **저장** 및 **열기** 단추를 둘 다 제공합니다.
Internet Explorer 10 이상에서 위의 샘플 코드를 실행하면 이 차이점을 보다 명확하게 확인할 수 있습니다.

기타 고급 기능

기타 파일 관련 고급 기능으로 파일 형식 필터링을 사용한 다중 파일 업로드(파일당 4GB)가 있습니다.

다음 예제에서는 사용자가 GIF(Graphics Interchange Format) 또는 JPEG 파일을 여러 개 선택할 수 있습니다.
HTML :
<input type="file" name="pic" multiple accept="image/gif, image/jpeg" />

자세한 내용은 HTML5(영문) 사양의 "파일 업로드 상태(type=file)" 섹션을 참조하세요.

로컬 파일을 읽는 방법

Internet Explorer 10 이상에서는 확장 또는 플러그 인 없이 로컬 파일 시스템(클라이언트 컴퓨터)에서 파일을 안전하게 읽을 수 있습니다.
이 항목에서는 다음과 같은 핵심 파일 관련 작업에 대해 설명합니다.
input 요소 및 DnD(끌어서 놓기) 상자를 사용하여 로컬 파일 선택. 선택한 파일 읽기, 선택한 파일 내용 표시, 파일 메타데이터(예: 크기, 파일 형식, 만든 날짜, 등) 표시. 화면에서 로컬 파일 시스템에 액세스하면 커다란 보안 허점이 있는 것처럼 보일 수도 있습니다.
하지만 많은 보호 기능(영문)이 포함되어 있습니다.
특히, 사용자가 권한을 제공한 경우에만 파일에 액세스할 수 있습니다.

참고 다음 코드 예제는 파일 API(영문)를 지원하는 브라우저(예: Internet Explorer 10 이상)가 필요합니다.

파일 선택

일반적으로 두 가지 방법으로 로컬 파일을 선택합니다.
가장 간단한 방법은 input 요소를 사용하는 것입니다.
다른 방법은 DnD 상자를 만드는 것으로 좀 더 복잡합니다.

입력 요소를 사용하여 파일 선택

input 요소에서 type="file"을 사용하면 상대적으로 간단하게 파일을 선택할 수 있습니다.

HTML :

```
<input type="file" id="fileSelector" multiple accept="image/*">
```

실행결과 : **찾아보기** 단추를 클릭할 때 multiple 특성을 사용하여 이후에 표시되는 파일 선택 대화 상자에서 여러 파일을 선택할 수 있습니다.
accept 특성을 사용하면 .jpg, .gif, .bmp, 등과 같은 그래픽 파일만 선택할 수 있습니다.
다음의 간단한 예에서는 input 요소를 사용하여 여러 그래픽 파일을 선택하는 방법을 보여줍니다.

예제 1

JavaScript :

```
<!DOCTYPE html>
<html>
  <head>
    <title>&lt;input&gt; File Selection</title>
    <meta http-equiv="X-UA-Compatible" content="IE=10">
  </head>
  <body>
```

```
<h1>HTML5 &lt;input&gt; File Selection</h1>
<h3>Example 1</h3>
<input type="file" id="fileSelector" multiple accept="image/*" /> <!-- By design, if you select the exact same files two or more times, the 'change' event will not fire. -->
<ul id="fileContentList" style="list-style-type: none;"></ul> <!-- This will be populated with <li> elements via JavaScript. -->
<script type="text/javascript">
  var message = [];

  if (!document.getElementById('fileSelector').files) {
    message = '<p>The ' +
              '<a  href="http://dev.w3.org/2006/webapi/FileAPI/"  target="_blank">File API</a>s ' +
              'are not fully supported by this browser.</p>' +
              '<p>Upgrade your browser to the latest version.</p>';

    document.querySelector('body').innerHTML = message;
  }
  else {
    document.getElementById('fileSelector').addEventListener('change', handleFileSelection, false); // Add an onchange event listener for the <input id="fileSelector"> element.
  }

  function handleFileSelection(evt) {
    var files = evt.target.files; // The files selected by the user (as a FileList object).

    if (!files) {
      msa.alert("<p>At least one selected file is invalid - do not select any folders.</p><p>Please reselect and try again.</p>");
      return;
    }

    // The variable "files" is an array of file objects.
    for (var i = 0, file; file = files[i]; i++) {
      var img_element = document.createElement('img'); // We've only allowed the user to select graphics files, so get ready to display them.
      img_element.src = window.URL.createObjectURL(file); // Assumes "file" is some sort of graphics file type.
```

```
            img_element.width = 150; // Make all images the same width.
            img_element.style.verticalAlign = "middle"; // Center the image in the middle of adjacent text.
            img_element.style.margin = "4px 4px 4px 0";
            img_element.onload = function() { window.URL.revokeObjectURL(this.src); } // The file URL is not needed once the file image has been fully loaded.

            var span_element = document.createElement('span');
            span_element.innerHTML = file.name;

            var li_element = document.createElement('li');
            li_element.appendChild(img_element);
            li_element.appendChild(span_element);

            document.getElementById('fileContentList').appendChild(li_element);
          } // for
        } // handleFileSelection
      </script>
      <script src="../utilities.js" type="text/javascript"></script> <!-- Provides the msa.alert() method. -->
    </body>
</html>
```

위에서 <meta http-equiv="X-UA-Compatible" content="IE=10">은 Windows Internet Explorer 에 페이지를 IE10 모드로 표시하도록 지시합니다.
자세한 내용은 문서 호환성 정의를 참조하세요.
script 블록을 body 블록의 끝에 배치하면 일반적으로 성능이 향상될 뿐만 아니라 이전에 렌더링된 DOM 요소(예:<input type="file" id="fileSelector" multiple accept="image/*" />)에 액세스할 수 있습니다.

예제 1 의 알고리즘은 간단합니다.
특정 파일 API 기능이 제공되지 않는 경우 필요한 파일 API 기능을 사용할 수 없다고 경고하는 HTML 로 모든 이전 태그를 교체합니다.
그렇지 않으면 input 요소에서 모든 변경 내용을 수신할 이벤트 수신기(handleFileSelection)를 추가합니다.
input 요소를 통해 파일을 하나 이상 선택하면 handleFileSelection 함수를 호출하는 change 이벤트가 발생됩니다.
handleFileSelection 에 전달되는 이벤트 개체에는 사용자가 선택한 파일 목록 즉, evt.target.files 가 포함되어 있습니다.

그런 다음 파일 목록의 각 파일 개체에 대해 이미지 미리 보기 목록을 만들고 연결된 파일 이름을 표시합니다.

참고 JavaScript 를 사용하는 Windows 스토어 앱에서는 alert 를 사용할 수 없으므로 msa.alert 를 대신 사용합니다. JavaScript 를 사용하는 Windows 스토어 앱에 대해 개발하는 경우가 아니면 msa.alert 를 JavaScript 의 기존 alert 로 교체해도 기능적인 손실이 발생하지 않습니다.

끌어서 놓기 상자를 사용하여 파일 선택

파일을 선택하는 이전의 input 요소 기술은 상대적으로 간단하지만 미적으로 만족하지 못할 수도 있습니다.
좀 더 정교한 방법으로 다음 예제와 같이 파일 DnD 상자를 만들 수 있습니다.

예제 2

JavaScript :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Drag &amp;amp; Drop File Selection</title>
    <meta http-equiv="X-UA-Compatible" content="IE=10">
    <style>
      #fileDropBox {
        width: 20em;
        line-height: 10em;
        border: 1px dashed gray;
        text-align: center;
        color: gray;
        border-radius: 7px;
    </style>
  </head>
  <body>
    <h1>HTML5 Drag and Drop File Selection</h1>
    <h3>Example 2</h3>
    <p>Using Windows Explorer (or similar), select one or more files (directories are not allowed), and then drag them to the below drop box:</p>
    <div id="fileDropBox">Drop files here.</div>
    <ul id="list"></ul>
    <script>
      var message = [];
```

```
    if (!window.FileReader) {
      message = '<p>The ' +
                '<a  href="http://dev.w3.org/2006/webapi/FileAPI/"  target="_blank">File
API</a>s ' +
                'are not fully supported by this browser.</p>' +
                '<p>Upgrade your browser to the latest version.</p>';

      document.querySelector('body').innerHTML = message;
    }
    else {
      // Set up the file drag and drop listeners:
      document.getElementById('fileDropBox').addEventListener('dragover',
handleDragOver, false);
      document.getElementById('fileDropBox').addEventListener('drop', handleFileSelection,
false);
    }

    function handleDragOver(evt) {
      evt.stopPropagation();  // Do not allow the dragover event to bubble.
      evt.preventDefault(); // Prevent default dragover event behavior.
    } // handleDragOver()

    function handleFileSelection(evt) {
      evt.stopPropagation(); // Do not allow the drop event to bubble.
      evt.preventDefault(); // Prevent default drop event behavior.

      var files = evt.dataTransfer.files; // Grab the list of files dragged to the drop box.

      if (!files) {
        msa.alert("<p>At least one selected file is invalid - do not select any
folders.</p><p>Please reselect and try again.</p>");
        return;
      }

      // "files" is a FileList of file objects. List a few file object properties:
      var output = [];
      for (var i = 0, f; i < files.length; i++) {
        try {
          f = files[i]; // If anything goes awry, the error would occur here.
```

```
            output.push('<li><strong>',
                        f.name, '</strong> (',
                        f.type || 'unknown file type',
                        ') - ',
                        f.size, ' bytes, last modified: ',
                        f.lastModifiedDate,
                        '</li>');
            document.getElementById('list').innerHTML = output.join('');
          } // try
          catch (fileError) {
            msa.alert("<p>An unspecified file error occurred.</p><p>Selecting one or more folders will cause a file error.</p>");
            console.log("The following error occurred at i = " + i + ": " + fileError); // Send the error object to the browser's debugger console window, if active.
            return;
          } // catch
        } // for
      } // handleFileSelection()
    </script>
    <script src="../utilities.js" type="text/javascript"></script> <!-- Provides the msa.alert() method. -->
  </body>
</html>
```

파일 DnD 상자는 CSS 스타일 div 요소(ID 는 fileDropBox)이며 두 개의 이벤트 수신기가 연결되어 있습니다.
첫 번째 이벤트 수신기인 handleDragOver 는 dragover 이벤트를 무효화하는 데 사용됩니다.
두 번째 이벤트 수신기인 handleFileSelection 은 파일 메타데이터 정보(파일 이름, 크기, 등)를 표시하는 데 사용됩니다.
handleFileSelection 함수를 호출하면 사용자가 선택한 파일 목록이 들어 있는 이벤트 개체(evt)가 전달됩니다.

JavaScript :

```
var files = evt.dataTransfer.files;
```

그런 다음 파일 목록에 있는 각 파일에 대해 연결된 파일 메타데이터 정보의 글머리 기호 목록(output 배열에 푸시됨)을 작성합니다.
이 글머리 기호 목록은 <ul id="list"> 요소를 사용하여 다음을 통해 표시됩니다.

JavaScript :

```
document.getElementById('list').innerHTML = output.join('');
```

파일 읽기

이제 파일을 선택할 수 있으므로 다음 단계에서는 선택한 텍스트 파일의 내용을 표시합니다. 다음 예제를 참조하세요.

예제 3

JavaScript :

```
<!DOCTYPE html>
<html>
  <head>
    <title>Drag &amp;amp; Drop File Selection</title>
    <meta http-equiv="X-UA-Compatible" content="IE=10">
    <style>
      #fileDropBox {
        width: 20em;
        line-height: 10em;
        border: 1px dashed gray;
        text-align: center;
        color: gray;
        border-radius: 7px;
    </style>
  </head>
  <body>
    <h1>HTML5 Drag and Drop File Selection</h1>
    <h3>Example 3</h3>
    <p>Using Windows Explorer (or similar), select one or more text files (directories are not allowed), and then drag them to the below drop box:</p>
    <div id="fileDropBox">Drop files here.</div>
    <script>
      var message = [];

      if (!window.FileReader) {
        message = '<p>The ' +
                  '<a  href="http://dev.w3.org/2006/webapi/FileAPI/"  target="_blank">File API</a>s ' +
                  'are not fully supported by this browser.</p>' +
                  '<p>Upgrade your browser to the latest version.</p>';
```

```
            document.querySelector('body').innerHTML = message;
        }
        else {
            // Set up the file drag and drop listeners:
            document.getElementById('fileDropBox').addEventListener('dragover',
handleDragOver, false);
            document.getElementById('fileDropBox').addEventListener('drop', handleFileSelection,
false);
        }

        function sanitizeHTML(htmlString) {
            var tmp = document.createElement('div');
            tmp.appendChild( document.createTextNode(htmlString) );
            return tmp.innerHTML;
        } // stripHtmlFromText

        function handleDragOver(evt) {
            evt.stopPropagation();   // Do not allow the dragover event to bubble.
            evt.preventDefault(); // Prevent default dragover event behavior.
        } // handleDragOver

        function displayFileText(evt) {
            var fileString = evt.target.result; // Obtain the file contents, which was read into
memory.

            //evt.target is a FileReader object, not a File object; so
window.URL.createObject(evt.target) won't work here!
            msa.alert("<pre>" + sanitizeHTML(fileString) + "</pre>", {width: 40, tile: true});   //
sanitizeHTML() is used in case the user selects one or more HTML or HTML-like files and the
<pre> tag preserves both spaces and line breaks.
        } // displayFileText

        function handleFileReadAbort(evt) {
            msa.alert("File read aborted.");
        } // handleFileReadAbort

        function handleFileReadError(evt) {
            var message;
```

```
        switch (evt.target.error.name) {
          case "NotFoundError":
            msa.alert("The file could not be found at the time the read was processed.");
            break;
          case "SecurityError":
            message = "<p>A file security error occured. This can be due to:</p>";
            message += "<ul><li>Accessing certain files deemed unsafe for Web applications.</li>";
            message += "<li>Performing too many read calls on file resources.</li>";
            message += "<li>The file has changed on disk since the user selected it.</li></ul>";
            msa.alert(message);
            break;
          case "NotReadableError":
            msa.alert("The file cannot be read. This can occur if the file is open in another application.");
            break;
          case "EncodingError":
            msa.alert("The length of the data URL for the file is too long.");
            break;
          default:
            msa.alert("File error code " + evt.target.error.name);
        } // switch
      } // handleFileReadError

      function startFileRead(fileObject) {
        var reader = new FileReader();

        // Set up asynchronous handlers for file-read-success, file-read-abort, and file-read-errors:
        reader.onloadend = displayFileText; // "onloadend" fires when the file contents have been successfully loaded into memory.
        reader.abort = handleFileReadAbort; // "abort" files on abort.
        reader.onerror = handleFileReadError; // "onerror" fires if something goes awry.

        if (fileObject) { // Safety first.
          reader.readAsText(fileObject); // Asynchronously start a file read thread. Other supported read methods include readAsArrayBuffer() and readAsDataURL().
```

```
        }
      } // startFileRead

      function handleFileSelection(evt) {
        evt.stopPropagation(); // Do not allow the drop event to bubble.
        evt.preventDefault(); // Prevent default drop event behavior.

        var files = evt.dataTransfer.files; // Grab the list of files dragged to the drop box.

        if (!files) {
          msa.alert("<p>At least one selected file is invalid - do not select any folders.</p><p>Please reselect and try again.</p>");
          return;
        }

        // "files" is a FileList of file objects. Try to display the contents of each file:
        for (var i = 0, file; file = files[i]; i++) {
          if (!file) {
            msa.alert("Unable to access " + file.name);
            continue; // Immediately move to the next file object.
          }
          if (file.size == 0) {
            msa.alert("Skipping " + file.name.toUpperCase() + " because it is empty.");
            continue;
          }
          if ( !file.type.match('text/.*') ) {
            msa.alert("Skipping " + file.name.toUpperCase() + " because it is not a known text file type.");
            continue;
          }
          startFileRead(file); // Asychronously fire off a file read request.
        } // for
      } // handleFileSelection
    </script>
    <script src="utilities.js" type="text/javascript"></script> <!-- Provides the msa.alert() method. -->
  </body>
</html>
```

예제 3 은 예제 2 를 기반으로 하므로 알고리즘이 비슷합니다.
특정 파일 API 기능이 제공되지 않는 경우 필요한 파일 API 기능을 사용할 수 없다고 경고하는 HTML 로 모든 이전 태그를 교체합니다.
그렇지 않으면 DnD 이벤트 수신기 두 개를 DnD 상자에 추가합니다.

JavaScript :
** Drop files here. **

파일을 Windows 탐색기 창 등에서 DnD 상자로 끌면 drop 이벤트가 발생하여 handleFileSelection 함수를 실행합니다.
handleFileSelection 에 전달되는 이벤트(evt)에는 사용자가 선택한 파일이 포함되어 있습니다.
handleFileSelection 은 evt.dataTransfer.files 을 통해 전달된 각 파일을 반복하면서 기본 오류 검사를 수행하고 startFileRead(file)를 통해 지정된 파일에 대한 비동기 읽기 요청을 실행합니다.
파일을 메모리로 읽은 다음 이벤트 처리기를 호출하여 파일 데이터에 유용한 몇 가지 작업을 수행합니다. 이에 대해서는 다음에 설명합니다.

startFileRead(fileObject) 함수는 전달되는 각 파일에 대해 새 FileReader 개체를 만듭니다. 이 파일별 FileReader 개체(코드의 reader)는 함수 포인터(예: 이벤트 처리기)를 포함할 속성을 표시합니다.
또한 reader 는 다양한 파일 읽기 메서드 특히, reader.readAsText 를 표시합니다. reasAsText 메서드가 파일 콘텐츠를 메모리로 비동기적으로 로드하면 reader.onloadend 에 포함된 함수 포인터가 호출됩니다.
즉, displayFileText 이벤트 처리기가 실행됩니다.
displayFileText 에 전달되는 이벤트 개체에는 reader.reasAsText 가 실시한 읽기 결과가 포함되어 있습니다. 더 정확히 말해서 evt.target.result 에는 연결된 텍스트 파일에 대해 요청된 읽기 결과가 문자열 형식으로 포함되어 있습니다
이 문자열(즉, 파일의 텍스트 내용)을 다음과 같이 표시합니다.

JavaScript :
msa.alert("<pre>" + sanitizeHTML(fileString) + "</pre>", {width: 40, tile: true});

사용자가 HTML 또는 HTML 형식 파일을 하나 이상 선택한 경우 santizeHTML 을 호출하여 "<"를 "<"로 ">"를 ">"로 바꿉니다. <pre> 태그는 파일 콘텐츠의 공백과 줄 바꿈을 유지하는 데 사용됩니다.
위에서 설명한 것처럼 표준 alert 메서드는 정상적으로 작동하지만 msa.alert 는 JavaScript 를 사용하는 Windows 스토어 앱에서는 금지됩니다.
msa.alert 에 대한 두 번째 매개 변수는 선택적 매개 변수를 이 메서드에 전달하는 개체 리터럴입니다.

특히, tile: true 는 여러 경고 상자를 바둑판식으로 배열(약간씩 오프셋 적용)하도록 허용하며 width: 40 은 40em 단위 너비인 경고 상자를 생성합니다.
startFileRead 함수로 돌아와서 파일 읽기 오류가 발생하거나 사용자가 읽기 작업을 취소할 경우 handleFileReadError 및 handleFileReadAbort 이벤트 처리기가 각각 호출됩니다.
이제 HTML5 및 JavaScript 를 사용하여 로컬 파일과 콘텐츠를 읽을 수 있습니다. 다른 FileReader 읽기 메서드도 시도해 보세요. 특히, readAsArrayBuffer 메서드를 사용하여 개발자 응용 프로그램을 작성해 보세요.

파일 API 실습

Step 1.

<input type="file" id="inputFile" multiple="multiple"/>

먼저 type 이 file 인 input tag 를 생성한다.
input tag 가 type 이 file 일 경우, 자동으로 탐색기와 연결된 버튼을 생성해준다.
탐색기를 열어서 파일을 선택하면 해당 파일들의 정보를 가지고 있는 FIleList 를 생성해준다.
파일을 탐색기에서 선택할 때 다수를 선택하게 할려면 multiple 속성을 주고, 한개씩만 선택하게 할려면 빼면 된다.

input tag 에서 파일을 선택하게 되면 FileList 객체가 내부적으로 생성이 된다.
FileList 는 File 객체의 배열이다. FileList 에서 File 의 접근은 직접 접근외에 별다는 메소드는 제공하지 않는다.

아래는 스펙에서 가져온 인터페이스 정보이다.

```
interface FileList {
    getter File? item(unsigned long index);
    readonly attribute unsigned long length;
};

interface File : Blob {
    readonly attribute DOMString name;
    readonly attribute Date lastModifiedDate;
};
```

Step 2. Javascript

FileList 에서는 File 을 찾는 것과 list 의 사이즈를 제공하며, File 에서는 이름과 마지막 수정 일자를 제공하는 것을 알 수가 있다.
추가적으로 스펙에는 나와있지 않지만 크롬 브라우저에서는 파일 사이즈도 알 수가 있다.

```
document.getElementById("inputFile").onchange=function(){
    var fileList=this.files;
//input tag 의 타입이 file 일 경우 내부적으로 files 라는 FileList 객체를 자동으로 생성합니다.

    var fileListSize=fileList.length;
    document.getElementById("fileListLength").setAttribute("value", fileListSize);
```

```
    var strOption='';
    for(var i=0; i<fileListSize; i++){
        var file=fileList[i];
        var fileName=file.name;
        var fileLastModifiedDate=file.lastModifiedDate;
        var fileSize=file.fileSize;

                        strOption+='<option>'+fileName+'('+fileSize+'  byte)  -
'+fileLastModifiedDate+'</option>';
    }

    document.getElementById("fileList").innerHTML=strOption;
};
```

onchange 이벤트가 일어날 때 자신이 선택한 FileList 를 가지고 와서 Select box 에 파일 명과 사이즈를 보여주는 함수이다.

HTML5 File API 구현

웹 브라우저(Gecko, Webkit 등..)상에서 로컬 영역의 특정 파일을 접근, 조작할 수 있는 여러가지 방법을 제공합니다.

단 제공되는 File Interface 는 로컬 보안상 파일 탐색기 또는 Drag & Drop 방식을 이용해 사용자가 직접 선택한 File 로 한정 시킵니다.

아래는 관련 소스를 모듈화 시킨 코드이며, 각 기능은 아래와 같습니다.

1. 생성자 함수:

File(form || undefined)

form: 해당 File 컨트롤을 포함하는 부모 엘리먼트(보통 Form 엘리먼트가 해당됨)

2. 각 File 컨트롤의 onchange 이벤트 등록

.change(callback);

callback: onchange 이벤트 시 핸들러

3. 각 Drop Zone 엘리먼트의 dragenter 이벤트 등록

.dragenter(elems, callback)

elems: 엘리먼트 or 엘리먼트 배열

callback: ondragenter 이벤트 시 핸들러

4. 각 Drop Zone 엘리먼트의 dragleave 이벤트 등록

dragleave(elems, callback)

elems: 엘리먼트 or 엘리먼트 배열
callback: dragleave 이벤트 시 핸들러

5. 각 Drop Zone 엘리먼트의 dragover 이벤트 등록
dragover(elems, callback)
elems: 엘리먼트 or 엘리먼트 배열
callback: dragover 이벤트 시 핸들러

6. 각 Drop Zone 엘리먼트의 dragdrop 이벤트 등록
dragdrop(elems, callback)
elems: 엘리먼트 or 엘리먼트 배열
callback: dragover 이벤트 시 핸들러

7. progress 관련 이벤트 등록
progress(data, before, update, load, dataType)
data: file 객체
before: 읽어드린 file 객체의 onloadstart 이벤트 시 핸들러
update: 읽어드린 file 객체의 onprogress 이벤트 시 핸들러
load: 읽어드린 file 객체의 onload 이벤트 시 핸들러
dataType: file 객체의 저장 데이터 타입

8. file 객체 반복자
File.each(files, callback, type)
files:컨트롤을 통해 선택된 파일 리스트
callback: 각 컨트롤의 반복자 callback
type: 허용 file 타입

9. 파일 읽기
Fil.read(data, callback, dataType)
data: 읽어드릴 file 객체
callback: 읽어드린 file 객체의 onload 이벤트 시 핸들러
dataType: file 객체의 저장 데이터 타입

10. Blob 생성
File.readBlob(data, start, end, callback)
data: 읽어드릴 file 객체
start: 시작 바이트 배열 index
end: 마지막 바이트 배열 index
callback: 읽어드린 file 객체의 onload 이벤트 시 핸들러

- 관련된 자세한 소스는 아래 jsbin.com 에 등록된 페이지를 살펴 보시기 바랍니다.

1. JSBin 소스 링크:

http://jsbin.com/urupic/2

2. 모듈 사용방법:

```
  window.onload = function (e) {
        //var target = document.getElementById('spin');
        //var spinner = new Spinner(opts).spin(target);

        var zone1 = document.getElementById('drop_zone1');
        var zone2 = document.getElementById('drop_zone2');
        var zone3 = document.getElementById('drop_zone3');

        var zones = [zone1];

        var progress = document.getElementById('progress_bar');
        var progress_percent = document.getElementById('progress_percent');

        File().change(function (e, files) {

            zone1.innerHTML = "Loading...";

            var that = this;
            var h = [];

            File.each(files, function (idx) {

                File.read(this, function (e, result) {

                    var html = '<li>';
                    html += '<div style="margin-top:20px;margin-left:10px;">';
                    html += '<img src="' + result + '" width="200" height="200" style="border:gray 2px solid" />';
                    html += '<div style="margin-top:10px"> name: ' + escape(this.name) + ' type: ' + this.type + ' size: ' + this.size + ', lastModifiedDate: ' + this.lastModifiedDate + ',' + this.lastModifiedDate.toLocaleDateString() + '</div>';
                    html += '<textarea style="margin-top:10px;width:600px;height:200px">' + result + '</textarea>';
```

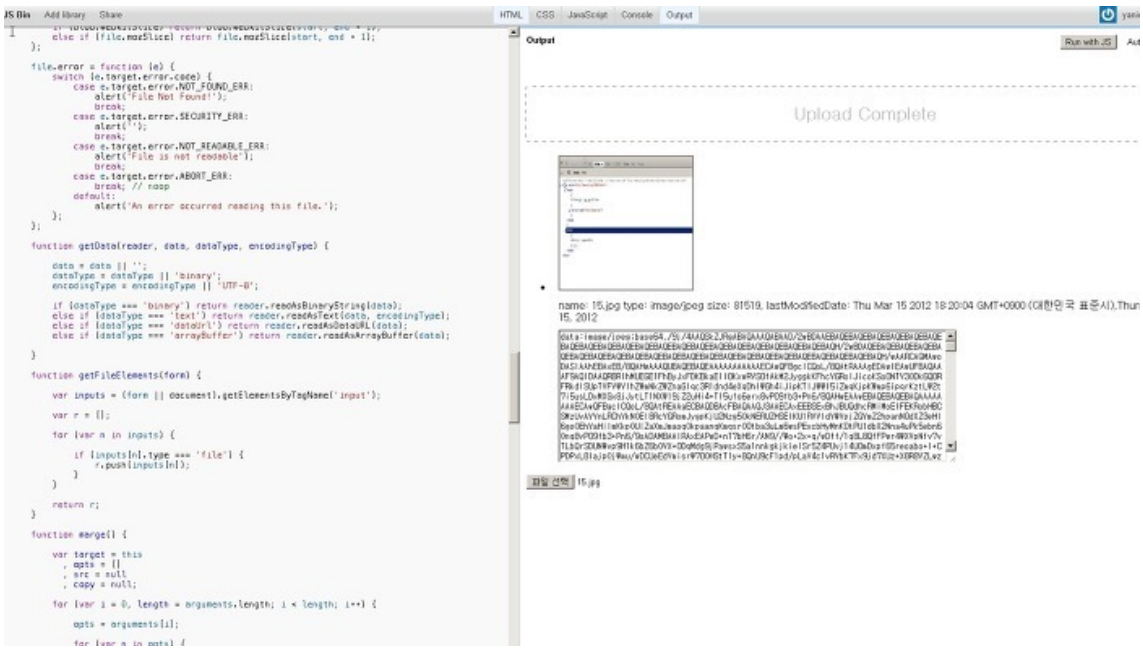
```
                    html += '</div>';
                    html += '</li>';

                    h.push(html);

                    if (idx >= files.length - 1) document.getElementById('image_list').innerHTML
+= h.join('');

                }, 'dataUrl');

                /*
                File.readBlob(this, 0, this.size, function (e, result) {
                });
                */

                that.progress(this, function (e) {
                    progress_percent.style.width = '0%';
                    progress_percent.textContent = '0%';

                    progress.className = 'loading';
                },
                            function (e, loaded, total) {

                                var percent = Math.round((loaded / total) * 100);

                                if (percent < 100) {
                                    progress_percent.style.width = percent + '%';
                                    progress_percent.textContent = percent + '%';
                                }
                            },
                            function (e) {
                                progress_percent.style.width = '100%';
                                progress_percent.textContent = '100%';

                                if (idx >= files.length - 1) {
                                    zone1.innerHTML = "Upload Complete";
                                    window.setTimeout(function () { progress.className = ''; },
2000);

                                }
```

```
                    });

            }, 'image');
        })
            .dragenter(zones, function (e, zone) {
                zone.style.backgroundColor = 'orange';
                zone.style.color = 'black';
            })
            .dragover(zones, function (e, zone) {
                zone.style.backgroundColor = 'orange';
                zone.style.color = 'black';
            })
            .dragleave(zones, function (e, zone) {
                zone.style.backgroundColor = 'white';
                zone.style.color = '#BBB';
            })
            .dragdrop(zones, function (e, files, zone) {

                zone.innerHTML = "Loading...";

                var that = this;
                var h = [];

                File.each(files, function (idx) {

                    File.read(this, function (e, result) {

                        var html = '<li>';
                        html += '<div style="margin-top:20px;margin-left:10px;">';
                        html += '<img src="' + result + '" width="200" height="200" style="border:gray 2px solid" />';
                        html += '<div style="margin-top:10px"> name: ' + escape(this.name) + ' type: ' + this.type + ' size: ' + this.size + ', lastModifiedDate: ' + this.lastModifiedDate + ',' + this.lastModifiedDate.toLocaleDateString() + '</div>';
                        html += '<textarea style="margin-top:10px;width:600px;height:200px">' + result + '</textarea>';
                        html += '</div>';
                        html += '</li>';
```

```
                    h.push(html);

                    if (idx >= files.length - 1) document.getElementById('image_list').innerHTML += h.join('');

                }, 'dataUrl');

                /*
                File.readBlob(this, 0, this.size, function (e, result) {
                });
                */

                that.progress(this, function (e) {
                    progress_percent.style.width = '0%';
                    progress_percent.textContent = '0%';

                    progress.className = 'loading';
                },
                    function (e, loaded, total) {

                        var percent = Math.round((loaded / total) * 100);

                        if (percent < 100) {
                            progress_percent.style.width = percent + '%';
                            progress_percent.textContent = percent + '%';
                        }
                    },
                    function (e, result) {
                        progress_percent.style.width = '100%';
                        progress_percent.textContent = '100%';

                        if (idx >= files.length - 1) {
                            zone.style.backgroundColor = 'white';
                            zone.style.color = '#BBB';
                            zone.innerHTML = "Upload Complete";

                            window.setTimeout(function () { progress.className = ''; }, 2000);

                        }
```

```
                    });

                }, 'image');
            });
    }

    function bind(elem, type, handler, capture) {

        type = typeof type === 'string' && type || '';
        handler = handler || function () { ; };

        if (elem.addEventListener) {
            elem.addEventListener(type, handler, capture);
        }
        else if (elem.attachEvent) {
            elem.attachEvent('on' + type, handler);
        }

        return elem;
    };
```

3. 실행화면 및 브라우저 지원 상황:



- **브라우저 지원 상황**:

File API - Working Draft

Method of manipulating file objects in web applications client-side, as well as programmatically selecting them and accessing their data.

*Usage stats:	Global
Support:	51.16%
Partial support:	4.18%
Total:	55.34%

Show all versions	IE	Firefox	Chrome	Safari	Opera	iOS Safari	Opera Mini	Android Browser
								2.1
		3.6				3.2		2.2
	7.0	12.0	19.0			4.0-4.1		2.3
	8.0	13.0	20.0	5.1		4.2-4.3		3.0
Current	9.0	14.0	21.0	6.0	12.0	5.0-5.1	5.0-7.0	4.0
Near future	10.0	15.0	22.0		12.5	6.0		
Farther future		16.0	23.0					

Sub-features: BlobBuilder API FileReader API Blob URLs

Notes | Known issues (0) | Resources (2) | Feedback

Microsoft is currently experimenting with the technology. Partial support in Safari refers to lacking FileReader support.

http://caniuse.com/#feat=fileapi