Read file content as data URL

INTRODUCTION TO A DATA URL

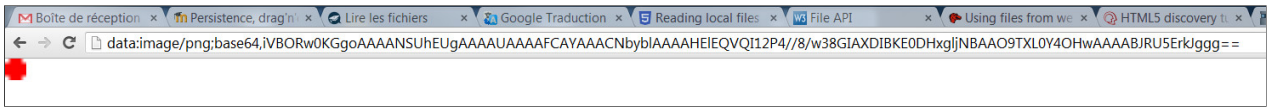
What is a data URL?

A data URL is a URL that includes type and content at the same time. It is useful, for example, for inlining images or videos in the HTML of a Web page (on mobile devices, this may speed up the loading of the page by reducing the number of HTTP requests).

Here is an example of a red square, as a data URL. Copy and paste it in the address bar of your browser, and you should see the red square:

```

```

This data URL in a browser address bar should look like that:



If we set the `src` attribute of an image element <img src="data:image/png...."> with the data URL of the above screenshot, it will work. Exactly as if you used a URL that started with `http://`

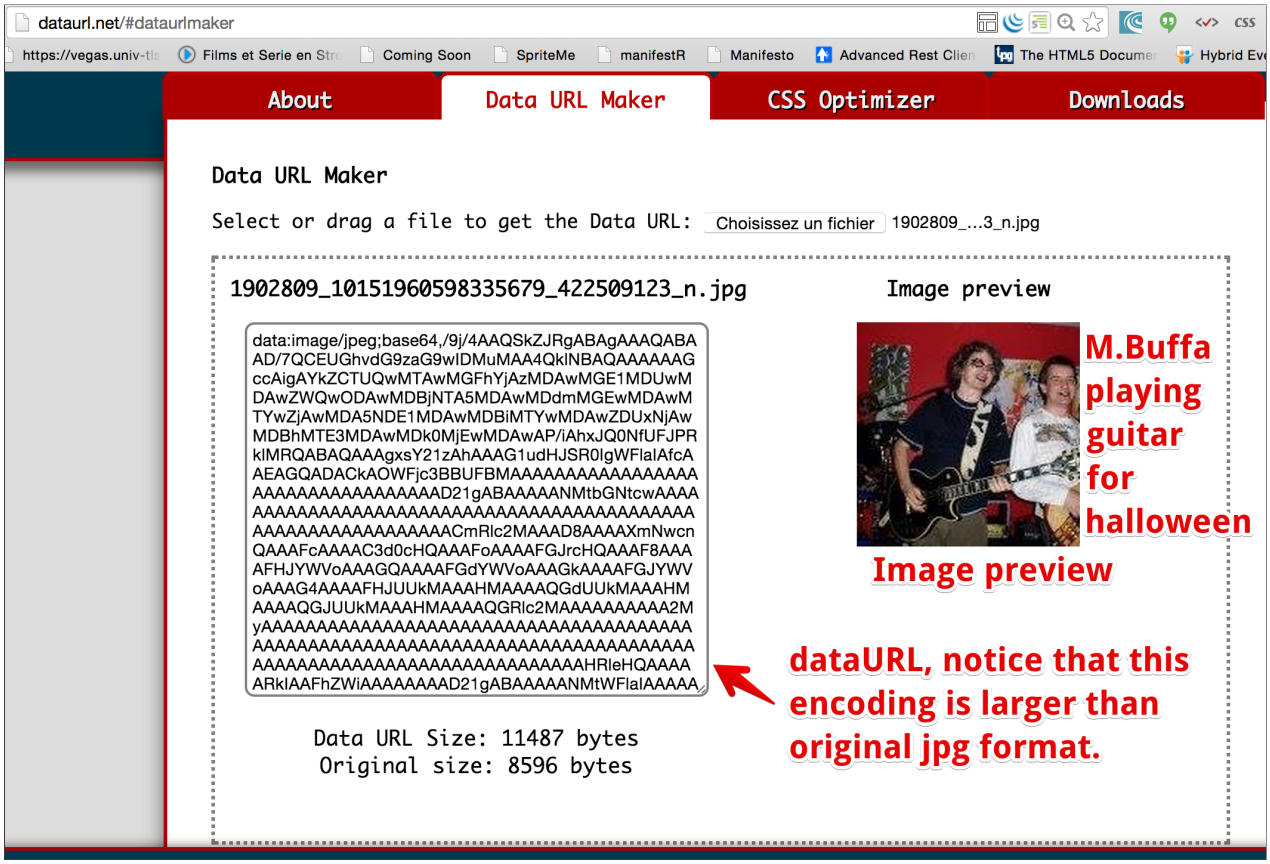
In your browser, you will see a small red circle rendered by this source code:

```
<imgsrc="
AAAFCAYAAACNbyblAAAAHElEQVQI12P4//8/w38GIAXDIBKE0DHxgljNBAAO
9TXL0Y4OHwAAAABJRU5ErkJggg==" alt="Red square" width=50 height=50/>
```

And here is the result:

This `dataURL` format enables storing a file content in a base64 format (as a string), and adds the MIME type specification of the content. The dataURL can therefore store a file as a URL readable with modern browsers. Its use is becoming more common on the Web, especially for mobile applications, as inlining images reduces the number of HTTP requests and makes Web page load faster.

You will find lots of Web sites and tools for generating dataURL from files, such as the DataURL maker Web site (screenshot below):



With the above example, you can copy and paste the characters on the left and use them with an `<img src="...">`. Just set the src attribute with it!

Notice that you can encode as `dataURL` any type of file, but the most frequent usage comes with media files (images, audio, video).

EXAMPLE 1: READ IMAGES AS DATA URL AND DISPLAY PREVIEWS IN THE PAGE

Example 1 is useful for forms that allow the user to select one or more pictures. Before sending the form, you might want to get a preview of the pictures in the HTML page. The`reader.readAsDataUrl` method is used for that.

Example on JS Bin or try it below in your browser:

Choose multiple images files: Choose Files No file chosen

Preview of selected images:

Source code extract:

```
 <label for="files">Choose multiple files:</label>
<input type="file" id="files" multiple
       onchange="readFilesAndDisplayPreview(this.files);"/><br/>
<p>Preview of selected images:</p>
<output id="list"></output>

<script>
 function readFilesAndDisplayPreview(files) {
   // Loop through the FileList and render image files as thumbnails.
```

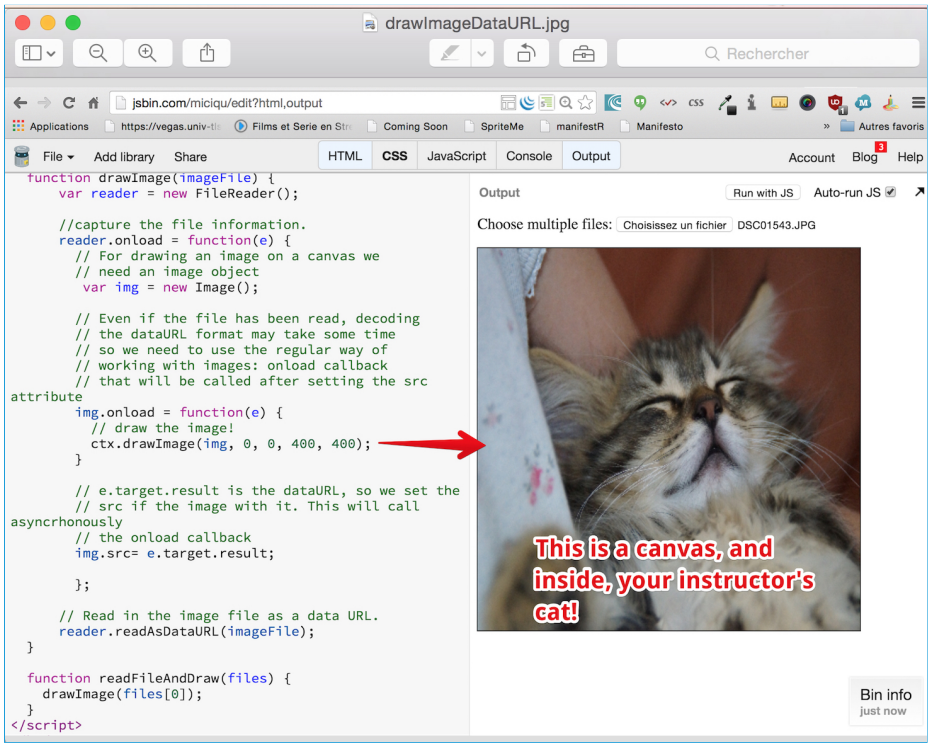
```
10.    for (var i = 0, f; f = files[i]; i++) {

       // Only process image files.
       if (!f.type.match('image.*')) {
           continue;
       }

       var reader = new FileReader();

       //capture the file information.
20.    reader.onload = function(e) {
           // Render thumbnail. e.target.result = the image content
           // as a data URL
          // create a span with CSS class="thumb", for nicer layout
          var span = document.createElement('span');
          // Add an img src=... in the span, with src= the dataURL of
          // the image
          span.innerHTML = "<img class='thumb' src='" +
                           e.target.result + "' alt='a picture'/>";
          // Insert the span in the output id=list
31.       document.getElementById('list').insertBefore(span, null);
        };
       // Read in the image file as a data URL.
       reader.readAsDataURL(f);
     }
   }
```

Explanations:

- *Line 35:* starts the reading of the file `f`. When `f` is read the`onload` callback will be called.
- *Lines 25-31:* we build, using the DOM API, a `<span class="thumb">...</span>` and inside we add an `<img src=the data url>` element with its `src` attribute equal to the url of the image that has been read (the image content as dataURL is in `e.target.result`). Finally, at line 31 we insert the span in the document before the current children of the `<output id="list">` element (declared at*line 5*).

EXAMPLE 2: READ A SINGLE LOCAL IMAGE FILE AND USE IT WITH DRAWIMAGE IN A CANVAS

Try it on JS Bin



Source code extract:

```
function drawImage(imageFile) {
  var reader = new FileReader();

  //capture the file information.
  reader.onload = function(e) {
    // For drawing an image on a canvas we
    // need an image object
    var img = new Image();
    // Even if the file has been read, decoding
    // the dataURL format may take some time
    // so we need to use the regular way of
    // working with images: onload callback
    // that will be called after setting the src attribute
    img.onload = function(e) {
      // draw the image!
      ctx.drawImage(img, 0, 0, 400, 400);
    }
    // e.target.result is the dataURL, so we set the
    // src if the image with it. This will call
    // asynchonously the onload callback
    img.src= e.target.result;
  };
  // Read in the image file as a data URL.
  reader.readAsDataURL(imageFile);
}

function readFileAndDraw(files) {
  drawImage(files[0]);
}
```

Explanations:

Remember how we worked with images on a canvas. We had to create an empty image object (*line 8*), set the `src` attribute of the image object (*line 23*), then use an `image.onload` callback (*line 15*), and we could only draw from inside the callback (*line 17*). This time, it's exactly the same, except that the URL comes from `e.target.result` in the `reader.onload` callback (*line 23*).

EXAMPLE 3 (ADVANCED): AN INSTAGRAM-LIKE PHOTO FILTER APPLICATION

Another very impressive example, has been developed by @GeorgianaB, a student of the first iteration of this course. This Web applications reads local image files, draws them into a canvas element and proposes different filters. This example is given "as is" for those of you who would like to go further. Just click on the link (or on the image below) and look at the source code.

Try this example online on gitHub (or click the screenshot below)