Hello World Walkthrough

Full source (https://github.com/mozilla/pdf.js/tree/master/examples/helloworld)

PDF.js heavily relies on the use of Promises (https://developer.mozilla.org/docs/Web/JavaScript/Reference/Global_Objects/Promise). If promises are new to you, it's recommended you become familiar with them before continuing on.

Document

The object structure of PDF.js loosely follows the structure of an actual PDF. At the top level there is a document object. From the document, more information and individual pages can be fetched. To get the document:

```
PDFJS.getDocument('helloworld.pdf')
```

Remember though that PDF.js uses promises, so the above will return a promise that is resolved with the document object.

```
PDFJS.getDocument('helloworld.pdf').then(function(pdf) {
  // you can now use *pdf* here
});
```

Page

Now that we have the document, we can get a page. Again, this uses promises.

```
pdf.getPage(1).then(function(page) {
  // you can now use *page* here
});
```

Rendering the Page

Each PDF page has its own viewport which defines the size in pixels(72DPI) and initial rotation. By default the viewport is scaled to the original size of the PDF, but this can be changed by modifying the viewport. When the viewport is created, an initial transformation matrix will also be created that takes into account the desired scale, rotation, and it transforms the coordinate system (the 0,0 point in PDF documents the bottom-left whereas canvas 0,0 is top-left).

```
var scale = 1.5;
var viewport = page.getViewport(scale);

var canvas = document.getElementById('the-canvas');
var context = canvas.getContext('2d');
canvas.height = viewport.height;
canvas.width = viewport.width;

var renderContext = {
  canvasContext: context,
  viewport: viewport
};
page.render(renderContext);
```

Alternatively, if you want the canvas to render to a certain pixel size you could do the following:

```
var desiredWidth = 100;
var viewport = page.getViewport(1);
var scale = desiredWidth / viewport.width;
var scaledViewport = page.getViewport(scale);
```

Interactive examples

Hello World with document load error handling

The example demonstrates how promises can be used to handle errors during loading. It also demonstrates how to wait until page loaded and rendered.

JavaScript HTML Result Edit in JSFiddle

```
// If absolute URL from the remote server is provided, configure the CORS
// header on that server.
var url = '//cdn.mozilla.net/pdfjs/helloworld.pdf';

// Disable workers to avoid yet another cross-origin issue (workers need
// the URL of the script to be loaded, and dynamically loading a cross-origin
// script does not work).
// PDFJS.disableWorker = true;

// The workerSrc property shall be specified.
PDFJS.workerSrc = '//mozilla.github.io/pdf.js/build/pdf.worker.js';

// Asynchronous download of PDF
var loadingTask = PDFJS.getDocument(url);
loadingTask.promise.then(function(pdf) {
  console.log('PDF loaded');

  // Fetch the first page
  var pageNumber = 1;
  pdf.getPage(pageNumber).then(function(page) {
    console.log('Page loaded');

    var scale = 1.5;
    var viewport = page.getViewport(scale);

    // Prepare canvas using PDF page dimensions
    var canvas = document.getElementById('the-canvas');
    var context = canvas.getContext('2d');
    canvas.height = viewport.height;
    canvas.width = viewport.width;

    // Render PDF page into canvas context
    var renderContext = {
      canvasContext: context,
      viewport: viewport
    };
    var renderTask = page.render(renderContext);
    renderTask.then(function () {
      console.log('Page rendered');
    });
  });
}, function (reason) {
  // PDF loading error
  console.error(reason);
});
```

Hello World using base64 encoded PDF

The PDF.js can accept any decoded base64 data as an array.

JavaScript HTML Result Edit in JSFiddle

JavaScript HTML Result Edit in JSFiddle

```
// atob() is used to convert base64 encoded PDF to binary-like data.
// (See also https://developer.mozilla.org/en-US/docs/Web/API/WindowBase64/
// Base64_encoding_and_decoding.)
var pdfData = atob(
  'JVBERi0xLjcKCjEgMCBvYmogICUgZW50cnkgcG9pbnQKPDwKICAvVHlwZSAvQ2F0YWxvZwog' +
  'IC9QYWdlcyAyIDAgUgo+PgplbmRvYmoKCjIgMCBvYmoKPDwKICAvVHlwZSAvUGFnZXMKICAv' +
  'TWVkaWFCb3ggWyAwIDAgMjAwIDIwMCBdCiAgL0NvdW50IDEKICAvS2lkcyBbIDMgMCBSIF0K' +
  'Pj4KZW5kb2JqCgozIDAgb2JqCjw8CiAgL1R5cGUgL1BhZ2UKICAvUGFyZW50IDIgMCBSCiAg' +
  'L1Jlc291cmNlcyA8PAogICAgL0ZvbnQgPDwKICAgICAgL0YxIDQgMCBSIAogICAgPj4KICA+' +
  'PgogIC9Db250ZW50cyA1IDAgUgo+PgplbmRvYmoKCjQgMCBvYmoKPDwKICAvVHlwZSAvRm9u' +
  'dAogIC9TdWJ0eXBlIC9UeXBlMQogIC9CYXNlRm9udCAvVGltZXMtUm9tYW4KPj4KZW5kb2Jq' +
  'Cgo1IDAgb2JqICAlIHBhZ2UgY29udGVudAo8PAogIC9MZW5ndGggNDQKPj4Kc3RyZWFtCkJU' +
  'CjcwIDUwIFRECi9GMSAxMiBUZgooSGVsbG8sIHdvcmxkISkgVGoKRVQKZW5kc3RyZWFtCmVu' +
  'ZG9iagoKeHJlZgowIDYKMDAwMDAwMDAwMCA2NTUzNSBmIAowMDAwMDAwMDEwIDAwMDAwIG4g' +
  'CjAwMDAwMDAwNzkgMDAwMDAgbiAKMDAwMDE3MyAwMDAwMCBuIAowMDAwMzAxIDAw' +
  'MDAwIG4gCjAwMDAwMDAzODAgMDAwMDAgbiAKdHJhaWxlcgo8PAogIC9TaXplIDYKICAvUm9v' +
  'dCAxIDAgUgo+PgpzdGFydHhyZWYKNDkyCiUlRU9G');

// Disable workers to avoid yet another cross-origin issue (workers need
// the URL of the script to be loaded, and dynamically loading a cross-origin
// script does not work).
// PDFJS.disableWorker = true;

// The workerSrc property shall be specified.
PDFJS.workerSrc = '//mozilla.github.io/pdf.js/build/pdf.worker.js';

// Using DocumentInitParameters object to load binary data.
var loadingTask = PDFJS.getDocument({data: pdfData});
loadingTask.promise.then(function(pdf) {
  console.log('PDF loaded');

  // Fetch the first page
  var pageNumber = 1;
  pdf.getPage(pageNumber).then(function(page) {
    console.log('Page loaded');

    var scale = 1.5;
    var viewport = page.getViewport(scale);

    // Prepare canvas using PDF page dimensions
    var canvas = document.getElementById('the-canvas');
    var context = canvas.getContext('2d');
    canvas.height = viewport.height;
    canvas.width = viewport.width;

    // Render PDF page into canvas context
    var renderContext = {
      canvasContext: context,
      viewport: viewport
    };
    var renderTask = page.render(renderContext);
    renderTask.then(function () {
      console.log('Page rendered');
```

```
    });
  });
}, function (reason) {
  // PDF loading error
  console.error(reason);
});
```

Previous/Next example

The same canvas cannot be used to perform to draw two pages at the same time – the example demonstrates how to wait on previous operation to be complete.

JavaScript HTML Result Edit in JSFiddle

```
// If absolute URL from the remote server is provided, configure the CORS
// header on that server.
var url = '//cdn.mozilla.net/pdfjs/tracemonkey.pdf';

// The workerSrc property shall be specified.
PDFJS.workerSrc = '//mozilla.github.io/pdf.js/build/pdf.worker.js';

var pdfDoc = null,
    pageNum = 1,
    pageRendering = false,
    pageNumPending = null,
    scale = 0.8,
    canvas = document.getElementById('the-canvas'),
    ctx = canvas.getContext('2d');

/**
 * Get page info from document, resize canvas accordingly, and render page.
 * @param num Page number.
 */
function renderPage(num) {
  pageRendering = true;
  // Using promise to fetch the page
  pdfDoc.getPage(num).then(function(page) {
    var viewport = page.getViewport(scale);
    canvas.height = viewport.height;
    canvas.width = viewport.width;

    // Render PDF page into canvas context
    var renderContext = {
      canvasContext: ctx,
      viewport: viewport
    };
    var renderTask = page.render(renderContext);

    // Wait for rendering to finish
    renderTask.promise.then(function() {
      pageRendering = false;
      if (pageNumPending !== null) {
        // New page rendering is pending
```

```
        renderPage(pageNumPending);
        pageNumPending = null;
      }
    });
  });

  // Update page counters
  document.getElementById('page_num').textContent = num;
}

/**
 * If another page rendering in progress, waits until the rendering is
 * finised. Otherwise, executes rendering immediately.
 */
function queueRenderPage(num) {
  if (pageRendering) {
    pageNumPending = num;
  } else {
    renderPage(num);
  }
}

/**
 * Displays previous page.
 */
function onPrevPage() {
  if (pageNum <= 1) {
    return;
  }
  pageNum--;
  queueRenderPage(pageNum);
}
document.getElementById('prev').addEventListener('click', onPrevPage);

/**
 * Displays next page.
 */
function onNextPage() {
  if (pageNum >= pdfDoc.numPages) {
    return;
  }
  pageNum++;
  queueRenderPage(pageNum);
}
document.getElementById('next').addEventListener('click', onNextPage);

/**
 * Asynchronously downloads PDF.
 */
PDFJS.getDocument(url).then(function(pdfDoc_) {
  pdfDoc = pdfDoc_;
  document.getElementById('page_count').textContent = pdfDoc.numPages;

  // Initial/first page rendering
  renderPage(pageNum);
});
```

```
});
```
