Extracting pages from a PDF document and saving them as separate image files, JavaScript edition with Promises

devblogs.microsoft.com/oldnewthing/20170627-00

June 27, 2017

Raymond Chen

Last time, we converted the C# version of the PDF Document sample program so that it saved the pages to disk as image files. Today we'll port those changes to JavaScript with Promises.

```
function viewPage() {
  WinJS.log && WinJS.log("", "sample", "status");

  var pageNumber = parseInt(pageNumberBox.value, 10);
  if (isNaN(pageNumber) || (pageNumber < 1) ||
    (pageNumber > pdfDocument.pageCount)) {
    WinJS.log && WinJS.log("Invalid page number.", "sample", "error");
    return;
  }

  output.src = "";
  progressControl.style.display = "block";

  // Convert from 1-based page number to 0-based page index.
  var pageIndex = pageNumber - 1;

  var page = pdfDocument.getPage(pageIndex);

  var picker = new Windows.Storage.Pickers.FileSavePicker();
  picker.fileTypeChoices["PNG image"] = [".png"];
  picker.pickSaveFileAsync().then(outfile => {
    if (outfile) {
      outfile.openTransactedWriteAsync().then(transaction => {
        var options = new PdfPageRenderOptions();
        options.destinationHeight = page.size.height * 2;
        options.destinationWidth = page.size.width * 2;
        page.renderToStreamAsync(transaction.stream, options).then(() => {
          transaction.close();
        });
      });
    }
  }).done(() => {
    page.close();
    // Delete the code that sets the blob into the image
    progressControl.style.display = "none";
  });
}
```

This is an uninspired direct translation of the C# code to JavaScript. We can imbue it with a little JavaScript inspiration by flattening the promise chain a bit.

```
var transaction;
var picker = new Windows.Storage.Pickers.FileSavePicker();
picker.fileTypeChoices["PNG image"] = [".png"];
picker.pickSaveFileAsync().then(outfile => {
  if (outfile) {
    return outfile.openTransactedWriteAsync();
  }
}).then(trans => {
  transaction = trans;
  if (transaction) {
      var options = new PdfPageRenderOptions();
      options.destinationHeight = page.size.height * 2;
      options.destinationWidth = page.size.width * 2;
      return page.renderToStreamAsync(transaction.stream, options);
  }
}).then(() => {
  transaction && transaction.close();
}).done(() => {
  page.close();
  // Delete the code that sets the blob into the image
  progressControl.style.display = "none";
});
```

Instead of nesting the promises, I chained them, and each step of the chain checks whether the previous step succeeded before proceeding. (If not, then that step does nothing.)

Alternatively, I could've thrown the Promise into an error state, but WinRT tries to reserve exceptions for unrecoverable errors, primarily out-of-memory conditions for a small allocation, or a programmer error. Errors that a program is expected to recover from are generally reported by an in-API mechanism. (There are notable exceptions to this principle, primarily in the I/O area.)

Anyway, you may have noticed that I used arrow functions, which are feature of ES6. Next time, I'm going to take it even further.

Raymond Chen

Follow