

Extracted from:

Modern Asynchronous JavaScript

Tackle Complex Async Tasks with Less Code

This PDF file contains pages extracted from *Modern Asynchronous JavaScript*, published by the Pragmatic Bookshelf. For more information or to purchase a paperback or PDF copy, please visit <http://www.pragprog.com>.

Note: This extract contains some colored text (particularly in code listing). This is available only in online versions of the books. The printed versions are black and white. Pagination might vary between the online and printed versions; the content is otherwise identical.

Copyright © 2021 The Pragmatic Programmers, LLC.

All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

The Pragmatic Bookshelf

Raleigh, North Carolina

The
Pragmatic
Programmers

Modern Asynchronous JavaScript

*Tackle Complex
Async Tasks
with Less Code*

Faraz K. Kelhini
edited by Margaret Eldridge

Modern Asynchronous JavaScript

Tackle Complex Async Tasks with Less Code

Faraz K. Kelhini

The Pragmatic Bookshelf

Raleigh, North Carolina



Many of the designations used by manufacturers and sellers to distinguish their products are claimed as trademarks. Where those designations appear in this book, and The Pragmatic Programmers, LLC was aware of a trademark claim, the designations have been printed in initial capital letters or in all capitals. The Pragmatic Starter Kit, The Pragmatic Programmer, Pragmatic Programming, Pragmatic Bookshelf, PragProg and the linking *g* device are trademarks of The Pragmatic Programmers, LLC.

Every precaution was taken in the preparation of this book. However, the publisher assumes no responsibility for errors or omissions, or for damages that may result from the use of information (including program listings) contained herein.

For our complete catalog of hands-on, practical, and Pragmatic content for software developers, please visit <https://pragprog.com>.

The team that produced this book includes:

CEO: Dave Rankin

COO: Janet Furlow

Managing Editor: Tammy Coron

Development Editor: Margaret Eldridge

Copy Editor: L. Sakhi MacMillan

Founders: Andy Hunt and Dave Thomas

For sales, volume licensing, and support, please contact support@pragprog.com.

For international rights, please contact rights@pragprog.com.

Copyright © 2021 The Pragmatic Programmers, LLC.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form, or by any means, electronic, mechanical, photocopying, recording, or otherwise, without the prior consent of the publisher.

ISBN-13: 978-1-68050-904-5

Encoded using the finest acid-free high-entropy binary digits.

Book version: P1.0—December 2, 2021

Making a User-Cancelable Async Request

When including large files on your page, you should take into account the fact that some users will be on limited bandwidth or mobile devices with expensive data plans. Therefore, the ability for a user to load and cancel loading large items is valuable.

Say you need to load a very large photo (in this case, 22 MB in size) from Wikipedia. You want to define a button that fetches the photo and another button that aborts the loading. Here's how the program will look:



You can see a live example of this program here:

https://eloux.com/async-js/examples/abort_ex08_complete.html

First, define an HTML `` element on the page. The `src` attribute of this element will be filled once the image is loaded. We also need an element to inform the user about the outcome, so define a `` element with a class of result. Next, create the buttons. We're going to disable the abort button until the load button is clicked, so give it a disabled attribute:

```
abort/abort_ex08.html
<!doctype html>
<html lang="en-us">

<head>
  <meta charset="utf-8">
  <title>Making a User Cancelable Async Request</title>
  <meta name="viewport" content="width=device-width, initial-scale=1">
```

```

    <script src="abort_ex08.js" defer></script>
  </head>

  <body>
    <image class="image">
    <span class="result"></span>
    <button class="loadBtn">Load Photo</button>
    <button class="abortBtn" disabled="disabled">Cancel Loading</button>
  </body>
</html>

```

Now, in the JavaScript file, we need to set up two functions: one to call when the Load Photo button is clicked and the other to call when the Cancel Loading button is clicked:

```

abort/abort_ex08.js
Line 1 // create a reference to each HTML element
- const loadBtn = document.querySelector('.loadBtn');
- const abortBtn = document.querySelector('.abortBtn');
- const image = document.querySelector('.image');
5 const result = document.querySelector('.result');
-
- const controller = new AbortController();
-
- // abort the request
10 abortBtn.addEventListener('click', () => controller.abort());
-
- // load the image
- loadBtn.addEventListener('click', async () => {
-   loadBtn.disabled = true;
15   abortBtn.disabled = false;
-
-   result.textContent = 'Loading...';
-
-   try {
20     const response = await fetch(`https://upload.wikimedia.org/wikipedia/com
- mons/a/a3/Kayakistas_en_Glaciard_Grey.jpg`, {signal: controller.signal});
-     const blob = await response.blob();
-     image.src = URL.createObjectURL(blob);
-
25     // remove the "Loading.." text
-     result.textContent = '';
-   }
-   catch (err) {
-     if (err.name === 'AbortError') {
30       result.textContent = 'Request successfully canceled';
-     } else {
-       result.textContent = 'An error occurred!'
-       console.error(err);
-     }
35   }
-
-

```

```
- loadBtn.disabled = false;  
- abortBtn.disabled = true;  
- });
```

Notice how line 13 of the code registers an async function to be called when the Load Photo button is clicked. Within the function, we disable the Load button to prevent another click and enable the Cancel Loading button. Next we attempt to retrieve the image using the standard `fetch()` function.

To be able to display the image we've retrieved, we need to convert it into an object URL. First use the `Blob()` constructor to get a Blob object (line 22). Now you can create a URL that refers to the Blob by passing the object into the `URL.createObjectURL()` method (line 23). All that's left to do to display the image is insert the resulting data into the `src` attribute of the image tag. At the end of the code, we revert the buttons to their original state.

What's a Blob?



Blob stands for *binary large object*, which is a data type containing a collection of binary data. In JavaScript, Blob serves as an essential data interchange method for several APIs. They're often used when working with data that isn't in a JavaScript-native format, such as images, audio, or other multimedia objects.

Now, what if we need to fetch multiple images and want to let the user abort them all at the same time?