
What is Resumable.js

Resumable.js is a JavaScript library providing multiple simultaneous, stable and resumable uploads via the [HTML5 File API](#).

The library is designed to introduce fault-tolerance into the upload of large files through HTTP. This is done by splitting each file into small chunks. Then, whenever the upload of a chunk fails, uploading is retried until the procedure completes. This allows uploads to automatically resume uploading after a network connection is lost either locally or to the server. Additionally, it allows for users to pause, resume and even recover uploads without losing state because only the currently uploading chunks will be aborted, not the entire upload.

Resumable.js does not have any external dependencies other than the [HTML5 File API](#). This is relied on for the ability to chunk files into smaller pieces. Currently, this means that support is widely available in to Firefox 4+, Chrome 11+, Safari 6+ and Internet Explorer 10+.

Samples and examples are available in the [samples/](#) folder. Please push your own as Markdown to help document the project.

How can I use it?

A new `Resumable` object is created with information of what and where to post:

```
1 var r = new Resumable({
2   target: '/api/photo/redeem-upload-token',
3   query: {upload_token: 'my_token'}
4 });
5 // Resumable.js isn't supported, fall back on a different method
6 if(!r.support) location.href = '/some-old-crappy-uploader';
```

To allow files to be selected and drag-dropped, you need to assign a drop target and a DOM item to be clicked for browsing:

```
1 r.assignBrowse(document.getElementById('browseButton'));
2 r.assignDrop(document.getElementById('dropTarget'));
```

It is recommended to use an HTML span for the browse button. Using an actual button does not work reliably across all browsers, because Resumable.js creates the file input as a child of this control, and this may be invalid in the case of an HTML button.

After this, interaction with Resumable.js is done by listening to events:

```
1 r.on('fileAdded', function(file, event){
2   ...
3 });
```

```
4 r.on('fileSuccess', function(file, message){
5     ...
6 });
7 r.on('fileError', function(file, message){
8     ...
9 });
```

How do I set it up with my server?

Most of the magic for Resumable.js happens in the user's browser, but files still need to be reassembled from chunks on the server side. This should be a fairly simple task, which can be achieved using any web framework or language that is capable of handling file uploads.

To handle the state of upload chunks, a number of extra parameters are sent along with all requests:

- **resumableChunkNumber**: The index of the chunk in the current upload. First chunk is 1 (no base-0 counting here).
- **resumableTotalChunks**: The total number of chunks.
- **resumableChunkSize**: The general chunk size. Using this value and **resumableTotalSize** you can calculate the total number of chunks. Please note that the size of the data received in the HTTP might be higher than **resumableChunkSize** for the last chunk for a file.
- **resumableTotalSize**: The total file size.
- **resumableIdentifier**: A unique identifier for the file contained in the request.
- **resumableFilename**: The original file name (since a bug in Firefox results in the file name not being transmitted in chunk multipart posts).
- **resumableRelativePath**: The file's relative path when selecting a directory (defaults to file name in all browsers except Chrome).

You should allow for the same chunk to be uploaded more than once; this isn't standard behaviour, but on an unstable network environment it could happen, and this case is exactly what Resumable.js is designed for.

For every request, you can confirm reception in HTTP status codes (can be changed through the **permanentErrors** option):

- 200, 201: The chunk was accepted and correct. No need to re-upload.
- 400, 404, 409, 415, 500, 501: The file for which the chunk was uploaded is not supported, cancel the entire upload.
- *Anything else*: Something went wrong, but try reuploading the file.

Handling GET (or test () requests)

Enabling the `testChunks` option will allow uploads to be resumed after browser restarts and even across browsers (in theory you could even run the same file upload across multiple tabs or different browsers). The `POST` data requests listed are required to use Resumable.js to receive data, but you can extend support by implementing a corresponding `GET` request with the same parameters:

- If this request returns a 200 HTTP code, the chunks is assumed to have been completed.
- If the request returns anything else, the chunk will be uploaded in the standard fashion. (It is recommended to return *204 No Content* in these cases if possible to avoid unwarranted notices in browser consoles.)

After this is done and `testChunks` enabled, an upload can quickly catch up even after a browser restart by simply verifying already uploaded chunks that do not need to be uploaded again.

Full documentation

Resumable

Configuration The object is loaded with a configuration hash:

```
1 var r = new Resumable({opt1:'val', ...});
```

All POST parameters can be omitted by setting them to a falsy value (e.g. `null`, `false` or empty string). Available configuration options are:

- `target` The target URL for the multipart POST request. This can be a `string` or a `function` that allows you to construct and return a value, based on supplied `params`. (Default: `/`)
- `testTarget` The target URL for the GET request to the server for each chunk to see if it already exists. This can be a `string` or a `function` that allows you to construct and return a value, based on supplied `params`. (Default: `null`)
- `chunkSize` The size in bytes of each uploaded chunk of data. The last uploaded chunk will be at least this size and up to two the size, see Issue #51 for details and reasons. (Default: `1*1024*1024`)
- `forceChunkSize` Force all chunks to be less or equal than `chunkSize`. Otherwise, the last chunk will be greater than or equal to `chunkSize`. (Default: `false`)
- `simultaneousUploads` Number of simultaneous uploads (Default: `3`)
- `fileParameterName` The name of the multipart request parameter to use for the file chunk (Default: `file`)
- `chunkNumberParameterName` The name of the chunk index (base-1) in the current upload POST parameter to use for the file chunk (Default: `resumableChunkNumber`)

-
- `totalChunksParameterName` The name of the total number of chunks POST parameter to use for the file chunk (Default: `resumableTotalChunks`)
 - `chunkSizeParameterName` The name of the general chunk size POST parameter to use for the file chunk (Default: `resumableChunkSize`)
 - `totalSizeParameterName` The name of the total file size number POST parameter to use for the file chunk (Default: `resumableTotalSize`)
 - `identifierParameterName` The name of the unique identifier POST parameter to use for the file chunk (Default: `resumableIdentifier`)
 - `fileNameParameterName` The name of the original file name POST parameter to use for the file chunk (Default: `resumableFilename`)
 - `relativePathParameterName` The name of the file's relative path POST parameter to use for the file chunk (Default: `resumableRelativePath`)
 - `currentChunkSizeParameterName` The name of the current chunk size POST parameter to use for the file chunk (Default: `resumableCurrentChunkSize`)
 - `typeParameterName` The name of the file type POST parameter to use for the file chunk (Default: `resumableType`)
 - `query` Extra parameters to include in the multipart request with data. This can be an object or a function. If a function, it will be passed a `ResumableFile` and a `ResumableChunk` object (Default: `{}`)
 - `testMethod` Method for chunk test request. (Default: `'GET'`)
 - `uploadMethod` HTTP method to use when sending chunks to the server (`POST`, `PUT`, `PATCH`) (Default: `POST`)
 - `parameterNamespace` Extra prefix added before the name of each parameter included in the multipart POST or in the test GET. (Default: `' '`)
 - `headers` Extra headers to include in the multipart POST with data. This can be an `object` or a `function` that allows you to construct and return a value, based on supplied `file` (Default: `{}`)
 - `method` Method to use when sending chunks to the server (`multipart` or `octet`) (Default: `multipart`)
 - `prioritizeFirstAndLastChunk` Prioritize first and last chunks of all files. This can be handy if you can determine if a file is valid for your service from only the first or last chunk. For example, photo or video meta data is usually located in the first part of a file, making it easy to test support from only the first chunk. (Default: `false`)
 - `testChunks` Make a GET request to the server for each chunks to see if it already exists. If implemented on the server-side, this will allow for upload resumes even after a browser crash or even a computer restart. (Default: `true`)
 - `preprocess` Optional function to process each chunk before testing & sending. Function is passed the chunk as parameter, and should call the `preprocessFinished` method on the

-
- chunk when finished. (Default: **null**)
- **preprocessFile** Optional function to process each file before testing & sending the corresponding chunks. Function is passed the file as parameter, and should call the **preprocessFinished** method on the file when finished. (Default: **null**)
 - **generateUniqueIdentifier(file, event)** Override the function that generates unique identifiers for each file. May return Promise-like object with **then()** method for asynchronous id generation. Parameters are the ES **File** object and the event that led to adding the file. (Default: **null**)
 - **maxFiles** Indicates how many files can be uploaded in a single session. Valid values are any positive integer and **undefined** for no limit. (Default: **undefined**)
 - **maxFilesErrorCallback(files, errorCount)** A function which displays the *please upload n file(s) at a time* message. (Default: displays an alert box with the message *Please n one file(s) at a time.*)
 - **minFileSize** The minimum allowed file size. (Default: **undefined**)
 - **minFileSizeErrorCallback(file, errorCount)** A function which displays an error a selected file is smaller than allowed. (Default: displays an alert for every bad file.)
 - **maxFileSize** The maximum allowed file size. (Default: **undefined**)
 - **maxFileSizeErrorCallback(file, errorCount)** A function which displays an error a selected file is larger than allowed. (Default: displays an alert for every bad file.)
 - **fileType** The file types allowed to upload. An empty array allow any file type. (Default: **[]**)
 - **fileTypeErrorCallback(file, errorCount)** A function which displays an error a selected file has type not allowed. (Default: displays an alert for every bad file.)
 - **maxChunkRetries** The maximum number of retries for a chunk before the upload is failed. Valid values are any positive integer and **undefined** for no limit. (Default: **undefined**)
 - **permanentErrors** List of HTTP status codes that define if the chunk upload was a permanent error and should not retry the upload. (Default: **[400, 404, 409, 415, 500, 501]**)
 - **chunkRetryInterval** The number of milliseconds to wait before retrying a chunk on a non-permanent error. Valid values are any positive integer and **undefined** for immediate retry. (Default: **undefined**)
 - **withCredentials** Standard CORS requests do not send or set any cookies by default. In order to include cookies as part of the request, you need to set the **withCredentials** property to true. (Default: **false**)
 - **xhrTimeout** The timeout in milliseconds for each request (Default: **0**)
 - **setChunkTypeFromFile** Set chunk content-type from original file.type. (Default: **false**, if **false** default Content-Type: **application/octet-stream**)
 - **dragOverClass** The class name to add on drag over an assigned drop zone. (Default: **dragover**)
-

Properties

- `.support` A boolean value indicator whether or not Resumable.js is supported by the current browser.
- `.opts` A hash object of the configuration of the Resumable.js instance.
- `.files` An array of `ResumableFile` file objects added by the user (see full docs for this object type below).

Methods

- `.assignBrowse(domNodes, isDirectory)` Assign a browse action to one or more DOM nodes. Pass in `true` to allow directories to be selected (Chrome only). See the note above about using an HTML span instead of an actual button.
- `.assignDrop(domNodes)` Assign one or more DOM nodes as a drop target.
- `.on(event, callback)` Listen for event from Resumable.js (see below)
- `.upload()` Start or resume uploading.
- `.pause()` Pause uploading.
- `.cancel()` Cancel upload of all `ResumableFile` objects and remove them from the list.
- `.progress()` Returns a float between 0 and 1 indicating the current upload progress of all files.
- `.isUploading()` Returns a boolean indicating whether or not the instance is currently uploading anything.
- `.addFile(file)` Add a HTML5 File object to the list of files.
- `.addFiles(files)` Add an Array of HTML5 File objects to the list of files.
- `.removeFile(file)` Cancel upload of a specific `ResumableFile` object on the list from the list.
- `.getFromUniqueIdentifier(uniqueIdentifier)` Look up a `ResumableFile` object by its unique identifier.
- `.getSize()` Returns the total size of the upload in bytes.

Events

- `.fileSuccess(file, message)` A specific file was completed. `message` is the response body from the server.
- `.fileProgress(file, message)` Uploading progressed for a specific file.
- `.fileAdded(file, event)` A new file was added. Optionally, you can use the browser `event` object from when the file was added.
- `.filesAdded(arrayAdded, arraySkipped)` New files were added (and maybe some have been skipped).

-
- `.fileRetry(file)` Something went wrong during upload of a specific file, uploading is being retried.
 - `.fileError(file, message)` An error occurred during upload of a specific file.
 - `.uploadStart()` Upload has been started on the Resumable object.
 - `.complete()` Uploading completed.
 - `.progress()` Uploading progress.
 - `.error(message, file)` An error, including `fileError`, occurred.
 - `.pause()` Uploading was paused.
 - `.beforeCancel()` Triggers before the items are cancelled allowing to do any processing on uploading files.
 - `.cancel()` Uploading was canceled.
 - `.chunkingStart(file)` Started preparing file for upload
 - `.chunkingProgress(file, ratio)` Show progress in file preparation
 - `.chunkingComplete(file)` File is ready for upload
 - `.catchAll(event, ...)` Listen to all the events listed above with the same callback function.

ResumableFile

Properties

- `.resumableObj` A back-reference to the parent `Resumable` object.
- `.file` The correlating HTML5 `File` object.
- `.fileName` The name of the file.
- `.relativePath` The relative path to the file (defaults to file name if relative path doesn't exist)
- `.size` Size in bytes of the file.
- `.uniqueIdentifier` A unique identifier assigned to this file object. This value is included in uploads to the server for reference, but can also be used in CSS classes etc when building your upload UI.
- `.chunks` An array of `ResumableChunk` items. You shouldn't need to dig into these.

Methods

- `.progress(relative)` Returns a float between 0 and 1 indicating the current upload progress of the file. If `relative` is `true`, the value is returned relative to all files in the Resumable.js instance.
- `.abort()` Abort uploading the file.

-
- `.cancel()` Abort uploading the file and delete it from the list of files to upload.
 - `.retry()` Retry uploading the file.
 - `.bootstrap()` Rebuild the state of a `ResumableFile` object, including reassigning chunks and XMLHttpRequest instances.
 - `.isUploading()` Returns a boolean indicating whether file chunks is uploading.
 - `.isComplete()` Returns a boolean indicating whether the file has completed uploading and received a server response.
 - `.markChunksCompleted()` starts upload from the next chunk number while marking all previous chunks complete. Must be called before `upload()` method.

Alternatives

This library is explicitly designed for modern browsers supporting advanced HTML5 file features, and the motivation has been to provide stable and resumable support for large files (allowing uploads of several GB files through HTTP in a predictable fashion).

If your aim is just to support progress indications during upload/uploading multiple files at once, `Resumable.js` isn't for you. In those cases, something like `Plupload` provides the same features with wider browser support.