Installation

You probably only need to look at the simple example(source) to get started. Continue reading for step by step instructions and different installation approaches.

Download the standalone dropzone.js and include it like this:

```
<script src="./path/to/dropzone.js"></script>
```

Dropzone is now activated and available as window.Dropzone .

Dropzone does not handle your file uploads on the server. You have to implement the code to receive and store the file yourself. See the section Server side implementation for more information.

This is all you need to get dropzone up and running, but if you want it to look like the dropzone on this page, you'll need to download dropzone.css from the dist folder.

With component

If you use component you can just add dropzone as a dependency:

"enyo/dropzone": "*"

Then include it like this:

```
var Dropzone = require("dropzone");
```

so it is activated and scans the document.

The basic CSS markup is also included with component, but if you want it to look the same as on this page, you have to download the CSS (see below).

With RequireJS

Dropzone is also available as an AMD module for RequireJS.

You can find the dropzone-amd-module in the downloads folder.

Usage

The typical way of using dropzone is by creating a form element with the class `dropzone`:

```
<form action="/file-upload"
      class="dropzone"
      id="my-awesome-dropzone"></form>
```

That’s it. Dropzone will find all form elements with the class dropzone, automatically attach itself to it, and upload files dropped into it to the specified `action` attribute. The uploaded files can be handled just as if there would have been a html input like this:

```
<input type="file" name="file" />
```

If you want another name than `file` you can configure dropzone with the option `paramName`.

If you’re using component don’t forget

to require("dropzone"); otherwise it won’ t be activated.

If you want your file uploads to work even without JavaScript, you can include an element with the class fallback that dropzone will remove if the browser is supported. If the browser isn’ t supported, Dropzone will not create fallback elements if there is a fallback element already provided. (Obviously, if the browser doesn’ t support JavaScript, the form will stay as is)

Typically this will look like this:

```
<form action="/file-upload" class="dropzone">
  <div class="fallback">
    <input name="file" type="file" multiple />
  </div>
</form>
```

Create dropzones programmatically

Alternatively you can create dropzones programmaticaly (even on non form elements) by instantiating the Dropzone class

```
// Dropzone class:
var myDropzone = new Dropzone("div#myId", { url: "/file/post"});
```

or if you use jQuery, you can use the jQuery plugin Dropzone ships with:

```
// jQuery
$("div#myId").dropzone({ url: "/file/post" });
```

Don’ t forget to specify an url option if you’ re not using a form element, since Dropzone doesn’ t know where to post to without an action attribute.

Server side implementation

Dropzone does not provide the server side implementation of handling the files, but the way files are uploaded is identical to simple file upload forms like this:

```
<form action="" method="post" enctype="multipart/form-data">
  <input type="file" name="file" />
</form>
```

To handle basic file uploads on the server, please look at the corresponding documentation. Here are a few documentations, if you think I should add some, please contact me.

- NodeJS with express
- Ruby on rails
- Complete PHP tutorial by startutorial.com
- Basic PHP file upload
- Tutorial for Dropzone and Lavarel (PHP) written by Maksim Surguy
- Symfony2 and Amazon S3
- File upload in ASP.NET MVC using Dropzone JS and HTML5

Paid documentations:

- eBook for Dropzone with PHP by startutorial.com.

Please look at the Dropzone FAQ if you need more information.

Configuration

There are two ways to configure dropzones.

The obvious way is to pass an options object when instantiating a dropzone programmatically like in the previous create dropzones programmatically section.

But if you just have HTML elements with the dropzone class, then you don’t programmatically instantiate the objects, so you have to store the configuration somewhere so Dropzone knows how to configure the dropzones when instantiating them.

This is done with the Dropzone.options object.

```
// "myAwesomeDropzone" is the camelized version of the HTML element's ID
Dropzone.options.myAwesomeDropzone = {
  paramName: "file", // The name that will be used to transfer the file
  maxFilesize: 2, // MB
  accept: function(file, done) {
    if (file.name == "justinbieber.jpg") {
      done("Naha, you don't.");
    }
    else { done(); }
  }
};
```

If you want to disable the auto discover behaviour of Dropzone, you can either disable it on a per element basis, or in general:

```
// Prevent Dropzone from auto discovering this element:
Dropzone.options.myAwesomeDropzone = false;
// This is useful when you want to create the
// Dropzone programmatically later

// Disable auto discover for all elements:
Dropzone.autoDiscover = false;
```

Configuration options

url	Has to be specified on elements other than form (or when the form doesn't have an action attribute). You can also provide a function that will be called with files and must return the url (since v3.12.0)
method	Defaults to "post" and can be changed to "put" if necessary. You can also provide a function that will be called with files and must return the method (since v3.12.0)
parallelUploads	How many file uploads to process in parallel (See theEnqueuing file uploads section for more info)
maxFilesize	in MB
filesizeBase	Defaults to 1000 . This defines whether a base of 1000 or 1024 should be used to calculate file sizes. 1000 is correct, because 1000 Bytes equal 1 Kilobyte , and 1024 Bytes equal 1 Kibibyte . You can change this to 1024 if you don't care about validity.
paramName	The name of the file param that gets transferred. Defaults to file . NOTE: If you have the option uploadMultiple set to true , then Dropzone will append [] to the name.
uploadMultiple	Whether Dropzone should send multiple files in one request. If this it set to true, then the fallback file input element will have the multiple attribute as well. This option will also trigger additional events (like processingmultiple). See the events section for more information.
headers	An object to send additional headers to the server. Eg: headers: { "My-Awesome-Header": "header value" }
addRemoveLinks	This will add a link to every file preview to remove or cancel (if already uploading) the file. The dictCancelUpload , dictCancelUploadConfirmation and dictRemoveFile options are used for the wording.
previewsContainer	defines where to display the file previews – if null the Dropzone element is used. Can be a plain HTMLElement or a CSS selector. The element should have the dropzone-previews class so the previews are displayed properly.
clickable	If true , the dropzone element itself will be clickable, if false nothing will be clickable. Otherwise you can pass an HTML element, a CSS selector (for multiple elements) or an array of those.

createImageThumbnails	
maxThumbnailFilesize	in MB. When the filename exceeds this limit, the thumbnail will not be generated.
thumbnailWidth	if null , the ratio of the image will be used to calculate it.
thumbnailHeight	the same as thumbnailWidth . If both are null, images will not be resized.
maxFiles	if not null defines how many files this Dropzone handles. If it exceeds, the event maxfilesexceeded will be called. The dropzone element gets the class dz-max-files-reached accordingly so you can provide visual feedback.
resize	is the function that gets called to create the resize information. It gets the file as first parameter and must return an object with srcX , srcY , srcWidth and srcHeight and the same for trg* . Those values are going to be used by ctx.drawImage() .
init	is a function that gets called when Dropzone is initialized. You can setup event listeners inside this function.
~~acceptedMimeTypes~~	Deprecated in favor of acceptedFiles
acceptedFiles	The default implementation of accept checks the file's mime type or extension against this list. This is a comma separated list of mime types or file extensions. Eg.: image/*,application/pdf,.psd . If the Dropzone is clickable this option will be used as accept parameter on the hidden file input as well.
accept	is a function that gets a file and a done function as parameter. If the done function is invoked without a parameter, the file will be processed. If you pass an error message it will be displayed and the file will not be uploaded. This function will not be called if the file is too big or doesn't match the mime types.
~~enqueueForUpload~~	Deprecated in favor of autoProcessQueue .
autoProcessQueue	When set to false you have to call myDropzone.processQueue() yourself in order to upload the dropped files. See below for more information on handling queues.
previewTemplate	is a string that contains the template used for each dropped image. Change it to fulfill your needs but make sure to properly provide all elements. You can include the HTML in your page in a <div id="preview-template" style="display: none;"></div> container, and set it like this: previewTemplate: document.querySelector('preview-template').innerHTML .

forceFallback	Defaults to false . If true the fallback will be forced. This is very useful to test your server implementations first and make sure that everything works as expected without dropzone if you experience problems, and to test how your fallbacks will look.
fallback	is a function that gets called when the browser is not supported. The default implementation shows the fallback input field and adds a text.
to translate dropzone, you can also provide these options:	
dictDefaultMessage	The message that gets displayed before any files are dropped. This is normally replaced by an image but defaults to "Drop files here to upload"
dictFallbackMessage	If the browser is not supported, the default message will be replaced with this text. Defaults to "Your browser does not support drag'n'drop file uploads."
dictFallbackText	This will be added before the file input files. If you provide a fallback element yourself, or if this option is null this will be ignored. Defaults to "Please use the fallback form below to upload your files like in the olden days."
dictInvalidFileType	Shown as error message if the file doesn't match the file type.
dictFileTooBig	Shown when the file is too big. {{filesize}} and {{maxFilesize}} will be replaced.
dictResponseError	Shown as error message if the server response was invalid. {{statusCode}} will be replaced with the servers status code.
dictCancelUpload	If addRemoveLinks is true, the text to be used for the cancel upload link.
dictCancelUploadConfirmation	If addRemoveLinks is true, the text to be used for confirmation when cancelling upload.
dictRemoveFile	If addRemoveLinks is true, the text to be used to remove a file.
dictMaxFilesExceeded	If maxFiles is set, this will be the error message when it's exceeded.

You can also overwrite all default event actions in the options. So if you provide the option drop you can overwrite the default drop event handler. You should be familiar with the code if you do that because you can easily break the upload like this. If you just want to do additional stuff, like adding a few classes here and there, listen to the events instead!

Enqueuing file uploads

When a file gets added to the dropzone, its status gets set to Dropzone.QUEUED (after the accept function check passes) which means that the file is now in the queue.

If you have the option autoProcessQueue set to true then the queue is immediately processed, after a file is dropped or an upload finished, by calling .processQueue() which checks how many files are currently uploading, and if it's less than options.parallelUploads , .processFile(file) is called.

If you set autoProcessQueue to false , then .processQueue() is never called implicitly. This means that you have to call it yourself when you want to upload all files currently queued.

Layout

The HTML that is generated for each file by dropzone is defined with the option previewTemplate which defaults to this:

```
<div class="dz-preview dz-file-preview">
  <div class="dz-details">
    <div class="dz-filename"><span data-dz-name></span></div>
    <div class="dz-size" data-dz-size></div>
    <img data-dz-thumbnail />
  </div>
  <div class="dz-progress"><span class="dz-upload" data-dz-uploadprogress></s
  <div class="dz-success-mark"><span>✔</span></div>
  <div class="dz-error-mark"><span>✘</span></div>
  <div class="dz-error-message"><span data-dz-errormessage></span></div>
</div>
```

The container (dz-preview) gets the dz-processing class when the file gets processed, dz-success when the file got uploaded and dz-error in case the file couldn’t be uploaded. In the latter case, data-dz-errormessage will contain the text returned by the server.

To overwrite the default template, use the previewTemplate config.

You can access the HTML of the file preview in any of the events with file.previewElement .

If you decide to rewrite the previewTemplate from scratch, you should put elements with the data-dz-* attributes inside:

- data-dz-name
- data-dz-size
- data-dz-thumbnail (This has to be an <img /> element and the alt and src attributes will be changed by Dropzone)
- data-dz-uploadprogress (Dropzone will change the style.width property from 0% to 100% whenever there’s a uploadprogress event)
- data-dz-errormessage

The default options for Dropzone will look for those element and update the content for it.

If you want some specific link to remove a file (instead of the built in addRemoveLinks config), you can simply insert elements in the template with the data-dz-remove attribute. Example:

```
<img src="removebutton.png" alt="Click me to remove the file." data-dz-remove /
```

You are not forced to use those conventions though. If you override all the default event listeners you can completely rebuild your layout from scratch.

See the installation section on how to add the stylesheet and spritemaps if you want your dropzone to look like the one on this page.

See the Theming section, for a more in depth look at how to change Dropzone’s UI.

I created an example where I made Dropzone look and feel exactly the way jQuery File Uploader does with a few lines of configuration code. Check it out!

Again, please look at the Dropzone FAQ if you’ re still unclear about some features.

Dropzone methods

If you want to remove an added file from the dropzone, you can call .removeFile(file) . This method also triggers the removedfile event.

Here’ s an example that would automatically remove a file when it’ s finished uploading:

```
myDropzone.on("complete", function(file) {
  myDropzone.removeFile(file);
});
```

If you want to remove all files, simply use .removeAllFiles() . Files that are in the process of being uploaded won’ t be removed. If you want files that are currently uploading to be canceled, call .removeAllFiles(true) which will cancel the uploads.

If you have autoProcessQueue disabled, you’ ll need to call .processQueue() yourself.

This can be useful if you want to display the files and let the user click an accept button to actually upload the file(s).

To access all files in the dropzone, use myDropzone.files .

To get

- all accepted files: `.getAcceptedFiles()`
- all rejected files: `.getRejectedFiles()`
- all queued files: `.getQueuedFiles()`
- all uploading files: `.getUploadingFiles()`

If you do not need a dropzone anymore, just call `.disable()` on the object. This will remove all event listeners on the element, and clear all file arrays. To reenable a Dropzone use `.enable()`.

If you don’t like the default browser modals for `confirm` calls, you can handle them yourself by overwriting `Dropzone.confirm`.

```
Dropzone.confirm = function(question, accepted, rejected) {
  // Ask the question, and call accepted() or rejected() accordingly.
  // CAREFUL: rejected might not be defined. Do nothing in that case.
};
```

Events

Dropzone triggers events when processing files, to which you can register easily, by calling `.on(eventName, callbackFunction)` on your instance.

Since listening to events can only be done on instances of Dropzone, the best place to setup your event listeners, is in the `init` function:

```
// The recommended way from within the init configuration:
Dropzone.options.myAwesomeDropzone = {
  init: function() {
    this.on("addedfile", function(file) { alert("Added file."); });
  }
};
```

If you create your Dropzones programmatically, you can setup your event listeners on your instances, like this:

```
// This example uses jQuery so it creates the Dropzone, only when the DOM has
// loaded.

// Disabling autoDiscover, otherwise Dropzone will try to attach twice.
Dropzone.autoDiscover = false;
// or disable for specific dropzone:
// Dropzone.options.myDropzone = false;

$(function() {
  // Now that the DOM is fully loaded, create the dropzone, and setup the
  // event listeners
  var myDropzone = new Dropzone("#my-dropzone");
  myDropzone.on("addedfile", function(file) {
    /* Maybe display some more file information on your page */
  });
})
```

This is a bit more complex, and not necessary unless you have a good reason to instantiate Dropzones programmatically.

Dropzone itself relies heavily on events. Everything that’s visual is created by listening to them. Those event listeners are setup in the default configuration of every Dropzone and can be overwritten, thus replacing the default behavior with your own event callback.

You should only do this when you really know how Dropzone works, and when you want to completely theme your Dropzone

Event list

Do not overwrite those as configuration options, unless you know what you're doing.

All of these receive the event as first parameter:	
drop	The user dropped something onto the dropzone
dragstart	The user started to drag anywhere
dragend	Dragging has ended
dragenter	The user dragged a file onto the Dropzone
dragover	The user is dragging a file over the Dropzone
dragleave	The user dragged a file out of the Dropzone
All of these receive the file as first parameter:	
addedfile	When a file is added to the list
removedfile	Called whenever a file is removed from the list. You can listen to this and delete the file from your server if you want to.
thumbnail	When the thumbnail has been generated. Receives thedataUrl as second parameter.
error	An error occured. Receives the errorMessage as second parameter and if the error was due to the XMLHttpRequest the xhr object as third.
processing	When a file gets processed (since there is a queue not all files are processed immediately). This event was called processingfile previously.
uploadprogress	Gets called periodically whenever the file upload progress changes. Gets the progress parameter as second parameter which is a percentage (0-100) and the bytesSent parameter as third which is the number of the bytes that have been sent to the server. When an upload finishes dropzone ensures that uploadprogress will be called with a percentage of 100 at least once. Warning: This function can potentially be called with the same progress multiple times.
sending	Called just before each file is sent. Gets the xhr object and the formData objects as second and third parameters, so you can modify them (for example to add a CSRF token) or add additional data.

success	The file has been uploaded successfully. Gets the server response as second argument. (This event was called finished previously)
complete	Called when the upload was either successful or erroneous.
canceled	Called when a file upload gets canceled.
maxfilesreached	Called when the number of files accepted reaches the maxFiles limit.
maxfilesexceeded	Called for each file that has been rejected because the number of files exceeds the maxFiles limit.
All of these receive a list of files as first parameter and are only called if the uploadMultiple option is true:	
processingmultiple	See processing for description.
sendingmultiple	See sending for description.
successmultiple	See success for description.
completemultiple	See complete for description.
canceledmultiple	See canceled for description.
Special events:	
totaluploadprogress	Called with the total uploadProgress (0-100), the totalBytes and the totalBytesSent . This event can be used to show the overall upload progress of all files.
reset	Called when all files in the list are removed and the dropzone is reset to initial state.
queuecomplete	Called when all files in the queue finish uploading.

Theming

If you want to theme your Dropzone to look fully customized, in most cases you can simply replace the preview HTML template, adapt your CSS, and maybe create a few additional event listeners.

You will go very far with this approach. As an example, I created an example where I made Dropzone look and feel exactly the way jQuery File Uploader does with a few lines of configuration code. Check it out!

As you can see, the biggest change is the previewTemplate . I then added a few additional event listeners to make it look exactly like the reference.

You can however, implement your UI completely from scratch.

Dropzone itself sets up a lot of event listeners when a Dropzone is created, that handle all your UI. They do stuff like: create a new HTML element, add the <img> element when provided with image data (with the thumbnail event), update the progress bar when the uploadprogress event fires, show a checkmark when the success event fires, etc…

Everything visual is done in those event handlers. If you would overwrite all of them with empty functions, Dropzone would still be fully functional, but wouldn' t display the dropped files anymore.

If you like the default look of Dropzone, but would just like to add a few bells and whistles here and there, you should just add additional event listeners instead.

Overwriting the default event listeners, and creating your own, custom Dropzone, would look something like this:

```
// This is an example of completely disabling Dropzone's default behavior.
// Do *not* use this unless you really know what you are doing.
Dropzone.myDropzone.options = {
  previewTemplate: document.querySelector('#template-container').innerHTML,
  // Specifing an event as an configuration option overwrites the default
  // `addedfile` event handler.
  addedfile: function(file) {
    file.previewElement = Dropzone.createElement(this.options.previewTemplate)
    // Now attach this new element some where in your page
  },
  thumbnail: function(file, dataUrl) {
    // Display the image in your file.previewElement
  },
  uploadprogress: function(file, progress, bytesSent) {
    // Display the progress
  }
  // etc...
};
```

Obviously this lacks the actual implementation. Look at the source to see how Dropzone does it internally.

You should use this option if you don’t need any of the default Dropzone UI, but are only interested in Dropzone for it’s event handlers, file upload and drag’n’drop functionality.

Tips

If you do not want the default message at all (»Drop files to upload (or click)«), you can put an element inside your dropzone element with the class `dz-message` and dropzone will not create the message for you.

Dropzone will submit any hidden fields you have in your dropzone form. So this is an easy way to submit additional data. You can also use the `params` option.

Dropzone adds data to the `file` object you can use when events fire. You can access `file.width` and `file.height` if it’s an image, as well as `file.upload` which is an object containing: `progress` (0-100), `total` (the total bytes) and `bytesSent`.

If you want to add additional data to the file upload that has to be specific for each file, you can register for the sending event:

```
myDropzone.on("sending", function(file, xhr, formData) {
  // Will send the filesize along with the file as POST data.
  formData.append("filesize", file.size);
  });
```

To access the preview html of a file, you can access file.previewElement . For example:

```
  myDropzone.on("addedfile", function(file) {
    file.previewElement.addEventListener("click", function() {
      myDropzone.removeFile(file);
    });
  });
```

If you want the whole body to be a Dropzone and display the files somewhere else you can simply instantiate a Dropzone object for the body, and define the previewsContainer option. The previewsContainer should have the dropzone-previews or dropzone class to properly display the file previews.

```
  new Dropzone(document.body, {
    previewsContainer: ".dropzone-previews",
    // You probably don't want the whole body
    // to be clickable to select files
    clickable: false
  });
```

Look at the github wiki for more examples.

If you have any problems using Dropzone, please try to find help on stackoverflow.com by using the dropzone.js tag. Only create an issue on Github when you think you found a bug or want to suggest a new feature.

Compatibility

This section describes compatibility with browsers and older versions of Dropzone.

Browser Support

- Chrome 7+
- Firefox 4+
- IE 10+
- Opera 12+ (Version 12 for MacOS is disabled because their API is buggy)
- Safari 6+

For all the other browsers, dropzone provides an oldschool file input fallback.

There is no workaround for drag’ n’ drop in older browsers - it simply isn’ t supported. The same goes for image previews, etc… But using dropzone, your users using an old browserwill be able to upload files. It just won’ t look and feel great. But hey, that’ s their fault.

Version 4.0

This is not a changelog. Only compatibility problems are listed.

- Changed the default `previewTemplate` . Check out the new one in the layout section.
- Using an already included SVG instead of a PNG spritemap (the CSS file is now the only additional file that you need to include)

Version 3.0

This is not a changelog. Only compatibility problems are listed.

- All classes are prefixed with `dz-` now to prevent clashing with other CSS definitions

- The way previewTemplate is defined has changed. You have to provide data-dz-* elements now
- If the server returns JSON, it will be parsed for error messages as well
- There’s a dict* option for all of the visible messages
- Lots of minor fixes and changes

Version 2.0

This is not a changelog. Only compatibility problems are listed.

Starting with version 2.0, Dropzone no longer depends on jQuery, but Dropzone still registers itself as a jQuery module if available.

That means that creating your Dropzones like this still works:

```
$("#my-dropzone").dropzone({ /* options */ });
```

If you create your Dropzones with the normal constructor though, you have to pass either the raw HTMLElement, or a selector string. So those versions all work:

```
// With jQuery
new Dropzone($("#my-dropzone").get(0));
// Without jQuery
new Dropzone("#my-dropzone");
new Dropzone(document.querySelector("#my-dropzone"));
```

Another thing that changed, is that Dropzone no longer stores its instances inside the element’s data property. So to get a dropzone for an element do this now:

```
// DEPRECATED, do not use:
```

```
$("#my-dropzone").data("dropzone"); // won't work anymore
// Do this now:
Dropzone.forElement(element); // Providing a raw HTMLElement
// or
Dropzone.forElement("#my-dropzone"); // Providing a selector string.
```