html form file download

HTML5 Contact Form With File Upload.

When the form is submitted, the javascript form submission event handler above collects the form data and sends it to the server side script.

The serverside script entry point is handler.php (see in your downloaded zip file). The script uses a library called FormHandler, which inturn, uses other libraries.

Here is the code of the handler.php.

You have to edit the handler.php and change "someone@gmail.com" to your email address. If you want to add a second or third email address, you can do so like this:

Server Side Form Validations.

For this form, the server side form validations are done using this PHPFormValidation library. See the documentation to add or update the validations.

HTML - Forms.

HTML Forms are required, when you want to collect some data from the site visitor. For example, during user registration you would like to collect information such as name, email address, credit card, etc.

A form will take input from the site visitor and then will post it to a back-end application such as CGI, ASP Script or PHP script etc. The back-end application will perform required processing on the passed data based on defined business logic inside the application.

There are various form elements available like text fields, textarea fields, drop-down menus, radio buttons, checkboxes, etc.

The HTML <form> tag is used to create an HTML form and it has following syntax −

Form Attributes.

Apart from common attributes, following is a list of the most frequently used form attributes −

Backend script ready to process your passed data.

Method to be used to upload data. The most frequently used are GET and POST methods.

Specify the target window or frame where the result of the script will be displayed. It takes values like _blank, _self, _parent etc.

You can use the enctype attribute to specify how the browser encodes the data before it sends it to the server. Possible values are −

application/x-www-form-urlencoded − This is the standard method most forms use in simple scenarios.

mutlipart/form-data − This is used when you want to upload binary data in the form of files like image, word file etc.

Note − You can refer to Perl & CGI for a detail on how form data upload works.

HTML Form Controls.

There are different types of form controls that you can use to collect data using HTML form −

Text Input Controls Checkboxes Controls Radio Box Controls Select Box Controls File Select boxes Hidden Controls Clickable Buttons Submit and Reset Button.

Text Input Controls.

There are three types of text input used on forms −

Single-line text input controls − This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

Password input controls − This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTMl <input> tag.

Multi-line text input controls − This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Single-line text input controls.

This control is used for items that require only one line of user input, such as search boxes or names. They are created using HTML <input> tag.

Example.

Here is a basic example of a single-line text input used to take first name and last name −

This will produce the following result −

Attributes.

Following is the list of attributes for <input> tag for creating text field.

Indicates the type of input control and for text input control it will be set to text .

Used to give a name to the control which is sent to the server to be recognized and get the value.

This can be used to provide an initial value inside the control.

Allows to specify the width of the text-input control in terms of characters.

Allows to specify the maximum number of characters a user can enter into the text box.

Password input controls.

This is also a single-line text input but it masks the character as soon as a user enters it. They are also created using HTML <input>tag but type attribute is set to password .

Example.

Here is a basic example of a single-line password input used to take user password −

This will produce the following result −

Attributes.

Following is the list of attributes for <input> tag for creating password field.

Indicates the type of input control and for password input control it will be set to password .

Used to give a name to the control which is sent to the server to be recognized and get the value.

This can be used to provide an initial value inside the control.

Allows to specify the width of the text-input control in terms of characters.

Allows to specify the maximum number of characters a user can enter into the text box.

Multiple-Line Text Input Controls.

This is used when the user is required to give details that may be longer than a single sentence. Multi-line input controls are created using HTML <textarea> tag.

Example.

Here is a basic example of a multi-line text input used to take item description −

This will produce the following result −

Attributes.

Following is the list of attributes for <textarea> tag.

Used to give a name to the control which is sent to the server to be recognized and get the value.

Indicates the number of rows of text area box.

Indicates the number of columns of text area box.

Checkbox Control.

Checkboxes are used when more than one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to checkbox. .

Example.

Here is an example HTML code for a form with two checkboxes −

This will produce the following result −

Attributes.

Following is the list of attributes for <checkbox> tag.

Indicates the type of input control and for checkbox input control it will be set to checkbox. .

Used to give a name to the control which is sent to the server to be recognized and get the value.

The value that will be used if the checkbox is selected.

Set to checked if you want to select it by default.

Radio Button Control.

Radio buttons are used when out of many options, just one option is required to be selected. They are also created using HTML <input> tag but type attribute is set to radio .

Example.

Here is example HTML code for a form with two radio buttons −

This will produce the following result −

Attributes.

Following is the list of attributes for radio button.

Indicates the type of input control and for checkbox input control it will be set to radio.

Used to give a name to the control which is sent to the server to be recognized and get the value.

The value that will be used if the radio box is selected.

Set to checked if you want to select it by default.

Select Box Control.

A select box, also called drop down box which provides option to list down various options in the form of drop down list, from where a user can select one or more options.

Example.

Here is example HTML code for a form with one drop down box.

This will produce the following result −

Attributes.

Following is the list of important attributes of <select> tag −

Used to give a name to the control which is sent to the server to be recognized and get the value.

This can be used to present a scrolling list box.

If set to "multiple" then allows a user to select multiple items from the menu.

Following is the list of important attributes of <option> tag −

The value that will be used if an option in the select box box is selected.

Specifies that this option should be the initially selected value when the page loads.

An alternative way of labeling options.

File Upload Box.

If you want to allow a user to upload a file to your web site, you will need to use a file upload box, also known as a file select box. This is also created using the <input> element but type attribute is set to file .

Example.

Here is example HTML code for a form with one file upload box −

This will produce the following result −

Attributes.

Following is the list of important attributes of file upload box −

Used to give a name to the control which is sent to the server to be recognized and get the value.

Specifies the types of files that the server accepts.

Button Controls.

There are various ways in HTML to create clickable buttons. You can also create a clickable button using <input>tag by setting its type attribute to button . The type attribute can take the following values −

This creates a button that automatically submits a form.

This creates a button that automatically resets form controls to their initial values.

This creates a button that is used to trigger a client-side script when the user clicks that button.

This creates a clickable button but we can use an image as background of the button.

Example.

Here is example HTML code for a form with three types of buttons −

This will produce the following result −

Hidden Form Controls.

Hidden form controls are used to hide data inside the page which later on can be pushed to the server. This control hides inside the code and does not appear on the actual page. For example, following hidden form is being used to keep current page number. When a user will click next page then the value of hidden control will be sent to the web server and there it will decide which page will be displayed next based on the passed current page.

Sending form data.

Once the form data has been validated on the client-side, it is okay to submit the form. And, since we covered validation in the previous article, we're ready to submit! This article looks at what happens when a user submits a form — where does the data go, and how do we handle it when it gets there? We also look at some of the security concerns associated with sending form data.

Prerequisites: Basic computer literacy, an understanding of HTML, and basic knowledge of HTTP and server-side programming. Objective: To understand what happens when form data is submitted, including getting a basic idea of how data is processed on the server.

First we'll discuss what happens to the data when a form is submitted.

Client/server architecture.

At it's most basic, the web uses a client/server architecture that can be summarized as follows. a client (usually a web browser) sends a request to a server (most of the time a web server like Apache, Nginx, IIS, Tomcat, etc.), using the HTTP protocol. The server answers the request using the same protocol.

An HTML form on a web page is nothing more than a convenient user-friendly way to configure an HTTP request to send data to a server. This enables the user to provide information to be delivered in the HTTP request.

Note : To get a better idea of how client-server architectures work, read our Server-side website programming first steps module.

On the client side: defining how to send the data.

The <form> element defines how the data will be sent. All of its attributes are designed to let you configure the request to be sent when a user hits a submit button. The two most important attributes are action and method .

The action attribute.

The action attribute defines where the data gets sent. Its value must be a valid relative or absolute URL. If this attribute isn't provided, the data will be sent to the URL of the page containing the form — the current page.

In this example, the data is sent to an absolute URL — https://example.com :

Here, we use a relative URL — the data is sent to a different URL on the same origin:

When specified with no attributes, as below, the <form> data is sent to the same page that the form is present on:

Note: It's possible to specify a URL that uses the HTTPS (secure HTTP) protocol. When you do this, the data is encrypted along with the rest of the request, even if the form itself is hosted on an insecure page accessed using HTTP. On the other hand, if the form is hosted on a secure page but you specify an insecure HTTP URL with the action attribute, all browsers display a security warning to the user each time they try to send data because the data will not be encrypted.

The names and values of the non-file form controls are sent to the server as name=value pairs joined with ampersands. The action value should be a file on the server that can handle the incoming data, including ensuring server-side validation. The server then responds, generally handling the data and loading the URL defined by the action attribute, causing a new page load (or a refresh of the existing page, if the action points to the same page).

How the data is sent depends on the method attribute.

The method attribute.

The method attribute defines how data is sent. The HTTP protocol provides several ways to perform a request; HTML form data can be transmitted via a number of different methods, the most common being the GET method and the POST method.

To understand the difference between those two methods, let's step back and examine how HTTP works. Each time you want to reach a resource on the Web, the browser sends a request to a URL. An HTTP request consists of two parts: a header that contains a set of global metadata about the browser's capabilities, and a body that can contain information necessary for the server to process the specific request.

The GET method.

The GET method is the method used by the browser to ask the server to send back a given resource: "Hey server, I want to get this resource." In this case, the browser sends an empty body. Because the body is empty, if a form is sent using this method the data sent to the server is appended to the URL.

Consider the following form:

Since the GET method has been used, you'll see the URL www.foo.com/?say=Hi&to=Mom appear in the browser address bar when you submit the form.

The data is appended to the URL as a series of name/value pairs. After the URL web address has ended, we include a question mark (?) followed by the name/value pairs, each one separated by an ampersand (&). In this case we are passing two pieces of data to the server:

say , which has a value of Hi to , which has a value of Mom.

The HTTP request looks like this:

Note : You can find this example on GitHub — see get-method.html (see it live also).

The POST method.

The POST method is a little different. It's the method the browser uses to talk to the server when asking for a response that takes into account the data provided in the body of the HTTP request: "Hey server, take a look at this data and send me back an appropriate result." If a form is sent using this method, the data is appended to the body of the HTTP request.

Let's look at an example — this is the same form we looked at in the GET section above, but with the method attribute set to POST .

When the form is submitted using the POST method, you get no data appended to the URL, and the HTTP request looks like so, with the data included in the request body instead:

The Content-Length header indicates the size of the body, and the Content-Type header indicates the type of resource sent to the server. We'll discuss these headers later on.

Note : You can find this example on GitHub — see post-method.html (see it live also).

Viewing HTTP requests.

HTTP requests are never displayed to the user (if you want to see them, you need to use tools such as the Firefox Network Monitor or the Chrome Developer Tools). As an example, your form data will be shown as follows in the Chrome Network tab. After submitting the form:

Open the developer tools. Select "Network" Select "All" Select "foo.com" in the "Name" tab Select "Headers"

You can then get the form data, as shown in the image below.

The only thing displayed to the user is the URL called. As we mentioned above, with a GET request the user will see the data in their URL bar, but with a POST request they won't. This can be very important for two reasons:

If you need to send a password (or any other sensitive piece of data), never use the GET method or you risk displaying it in the URL bar, which would be very insecure. If you need to send a large amount of data, the POST method is preferred because some browsers limit the sizes of URLs. In addition, many servers limit the length of URLs they accept.

On the server side: retrieving the data.

Whichever HTTP method you choose, the server receives a string that will be parsed in order to get the data as a list of key/value pairs. The way you access this list depends on the development platform you use and on any specific frameworks you may be using with it.

Example: Raw PHP.

PHP offers some global objects to access the data. Assuming you've used the POST method, the following example just takes the data and displays it to the user. Of course, what you do with the data is up to you. You might display it, store it into a database, send it by email, or process it in some other way.

This example displays a page with the data we sent. You can see this in action in our example php-example.html file — which contains the same example form as we saw before, with a method of POST and an action of php-example.php . When it is submitted, it sends the form data to php-example.php, which contains the PHP code seen in the above block. When this code is executed, the output in the browser is Hi Mom .

Note : This example won't work when you load it into a browser locally — browsers cannnot interpret PHP code, so when the form is submitted the browser will just offer to download the PHP file for you. To get it to work, you need to run the example through a PHP server of some kind. Good options for local PHP testing are MAMP (Mac and Windows) and AMPPS (Mac, Windows, Linux).

Note also that if you are using MAMP but don't have MAMP Pro installed (or if the MAMP Pro demo time trial has expired), you might have trouble getting it working. To get it working again, we have found that you can load up the MAMP app, then choose the menu options MAMP > Preferences > PHP , and set "Standard Version:" to "7.2.x" (x will differ depending on what version you have installed).

Example: Python.

This example shows how you would use Python to do the same thing — display the submitted data on a web page. This uses the Flask framework for rendering the templates, handling the form data submission, etc. (see python-example.py).

The two templates referenced in the above code are as follows (these need to be in a subdirectory called templates in the same directory as the python-example.py file, if you try to run the example yourself):

: The same form as we saw above in the The POST method section but with the action set to > . This is a Jinja2 template, which is basically HTML but can contain calls to the Python code that is running the web server contained in curly braces. url_for('hello') is basically saying "redirect to /hello when the form is submitted". : This template just contains a line that renders the two bits of data passed to it when it is rendered. This is done via the hello() function seen above, which runs when the /hello URL is navigated to.

Note : Again, this code won't work if you just try to load it into a browser directly. Python works a bit differently to PHP — to run this code locally you'll need to install Python/PIP, then install Flask using pip3 install flask . At this point you should be able to run the example using python3 python-example.py , then navigating to localhost:5000 in your browser.

Other languages and frameworks.

There are many other server-side technologies you can use for form handling, including Perl, Java, .Net, Ruby, etc. Just pick the one you like best. That said, it's worth noting that it's very uncommon to use these technologies directly because this can be tricky. It's more common to use one of the many high quality frameworks that make handling forms easier, such as:

for Python (a bit more heavyweight than Flask, but with more tools and options). for Node.js. for PHP. for Ruby.

It's worth noting that even using these frameworks, working with forms isn't necessarily easy . But it's much easier than trying to write all the functionality yourself from scratch, and will save you a lot of time.

Note : It is beyond the scope of this article to teach you any server-side languages or frameworks. The links above will give you some help, should you wish to learn them.

A special case: sending files.

Sending files with HTML forms is a special case. Files are binary data — or considered as such — whereas all other data is text data. Because HTTP is a text protocol, there are special requirements for handling binary data.

The enctype attribute.

This attribute lets you specify the value of the Content-Type HTTP header included in the request generated when the form is submitted. This header is very important because it tells the server what kind of data is being sent. By default, its value is application/x-www-form-urlencoded . In human terms, this means: "This is form data that has been encoded into URL parameters."

If you want to send files, you need to take three extra steps:

Set the method attribute to POST because file content can't be put inside URL parameters. Set the value of enctype to multipart/form-data because the data will be split into multiple parts, one for each file plus one for the text data included in the form body (if text is also entered into the form). Include one or more <input type="file"> controls to allow your users to select the file(s) that will be uploaded.

Note: Servers can be configured with a size limit for files and HTTP requests in order to prevent abuse.

Security issues.

Each time you send data to a server, you need to consider security. HTML forms are by far the most common server attack vectors (places where attacks can occur). The problems never come from the HTML forms themselves — they come from how the server handles data.

The Website security article of our server-side learning topic discusses a number of common attacks and potential defenses against them in detail. You should go and check that article out, to get an idea of what's possible.

Be paranoid: Never trust your users.

So, how do you fight these threats? This is a topic far beyond this guide, but there are a few rules to keep in mind. The most important rule is: never ever trust your users, including yourself; even a trusted user could have been hijacked.

All data that comes to your server must be checked and sanitized. Always. No exception.

Escape potentially dangerous characters . The specific characters you should be cautious with vary depending on the context in which the data is used and the server platform you employ, but all server-side languages have functions for this. Things to watch out for are character sequences that look like executable code (such as JavaScript or SQL commands). Limit the incoming amount of data to allow only what's necessary . Sandbox uploaded files . Store them on a different server and allow access to the file only through a different subdomain or even better through a completely different domain.

You should avoid many/most problems if you follow these three rules, but it's always a good idea to get a security review performed by a competent third party. Don't assume that you've seen all the possible problems.

Summary.

As we'd alluded to above, sending form data is easy, but securing an application can be tricky. Just remember that a front-end developer is not the one who should define the security model of the data.It's possible to perform client-side form validation, but the server can't trust this validation because it has no way to truly know what has really happened on the client-side.

If you've worked your way through these tutorials in order, you now know how to markup and style a form, do client-side validation, and have some idea about submitting a form.

See also.

If you want to learn more about securing a web application, you can dig into these resources:

Html form file download.

The HTML method of uploading files into the Document Manager is the most basic method. It has the advantage of allowing you to transfer files to and from the Document Manager without having to install a third-party plug-in. It allows single-file selection for uploads and downloads, and some drawing reference file handling. It does not support uploading or downloading folders, shortcuts, or empty documents.

To upload files using the basic (HTML) file transfer method.

1 In the project or shell or Company Documents node, folders view, select the destination (target) folder into which you want to upload the files. 2 Click the Upload button or select File > Upload . The Upload Files and Folders window opens. 3 Click the Browse button and select the file to upload. 4 You can add some properties information at this time in the File Properties box: document title, revision number, and issue date. This information is optional. This information can also be added or imported at a later time. 5 To add another file to upload, click Add Row and select the next file to upload. If you want to remove a file from the list, select the row and click Remove . 6 For more options, click the Advanced Options button at the bottom of the window. The File Upload window expands to reveal additional options for handling drawing and reference files and files of the same name. 7 Click OK to upload the files into the selected destination folder. A progress indicator will display the transfer rate and progress of the upload. 8 When the upload is complete, the completion summary will display. The time it takes to upload depends upon the number and size of the files. If any files failed to upload, for example, duplicate file names, they will be listed on the summary. Click Close to close the window.

Note: If you have uploaded drawing and reference files marked as missing, see "Resolving Missing Reference Files (Reference Manager)".

Bootstrap File Input.

Bootstrap File Input is a field which the user can use to upload one or more files (photos, documents or any other file type) from local storage.

Standard file inputs usually leave a lot to be desired in terms of design, but MDB takes care of that by enhancing them with Material Design best practices.

Some of the most common use examples are:

CSV upload to CRM system Avatar picture upload Simple GIF upload.

Below you can see a number of Bootstrap file inputs created with Material Design for Bootstrap.

Note info: The recommended plugin to animate default file input: bs-custom-file-input, that’s what we are using currently in our packages and you don't have to add by yourself.

Note info: If you need more advanced functionalities, have a look at our Drag and drop file upload plugin.

Default file input.

Default styling for Bootstrap File Input component.

The file input is the most gnarly of the bunch and requires additional JavaScript if you’d like to hook them up with functional Choose file… and selected file name text.