This feature lets people upload both text and binary files. With PHP's authentication and file manipulation functions, you have full control over who is allowed to upload and what is to be done with the file once it has been uploaded.

PHP is capable of receiving file uploads from any RFC-1867 compliant browser.

PHP also supports PUT-method file uploads as used by Netscape Composer and W3C's Amaya clients. See the [PUT Method Support](http://php.net/manual/en/features.file-upload.put-method.php) for more details.

Example #1 File Upload Form

A file upload screen can be built by creating a special form which looks something like this:

<!DOCTYPE html>

<html>

<head>

<title>Upload your files</title>

</head>

<body>

<form enctype="multipart/form-data" action="upload.php" method="POST">

<p>Upload your file</p>

<input type="file" name="uploaded\_file"></input><br />

<input type="submit" value="Upload"></input>

</form>

</body>

</html>

<?PHP

if(!empty($\_FILES['uploaded\_file']))

{

$path = "uploads/";

$path = $path . basename( $\_FILES['uploaded\_file']['name']);

if(move\_uploaded\_file($\_FILES['uploaded\_file']['tmp\_name'], $path)) {

echo "The file ". basename( $\_FILES['uploaded\_file']['name']).

" has been uploaded";

} else{

echo "There was an error uploading the file, please try again!";

}

}

?>

The MAX\_FILE\_SIZE hidden field (measured in bytes) must precede the file input field, and its value is the maximum filesize accepted by PHP. This form element should always be used as it saves users the trouble of waiting for a big file being transferred only to find that it was too large and the transfer failed. Keep in mind: fooling this setting on the browser side is quite easy, so never rely on files with a greater size being blocked by this feature. It is merely a convenience feature for users on the client side of the application. The PHP settings (on the server side) for maximum-size, however, cannot be fooled.