

# Uploader API

## Uploader Overview

The Infusion Uploader provides an easy way for users to upload many files at once, providing useful feedback about the status and progress of each file along the way.

Uploader implements several different ways of to upload multiple files, including the new HTML 5 supported added in Infusion 1.3. With its built-in support for progressive enhancement, users will automatically receive a version of the Uploader best suited to the capabilities of their browsers. There are three different flavours of Uploader:

1. Single file: delivered to browsers that don't support JavaScript, Flash or HTML 5
2. Flash with [SWFUpload](#): delivered to older browsers with JavaScript enabled (especially IE)
3. HTML 5: the best and most widely-supported version of Uploader, suitable for modern browsers

The HTML 5 version of the Uploader will be delivered to modern, standards-compliant browsers, including:

- Firefox 3.6 and higher
- Safari 4 and higher
- Google Chrome

Internet Explorer users will receive the Flash version of the Uploader. Due to limitations in Flash, this version of the Uploader provides a less polished user experience and limited support for accessibility. As a result, we encourage you and your users to upgrade to an HTML 5-compatible browser such as Firefox 3.6+, Safari 4+, or Chrome.

**NOTE:** If you will be testing the Flash version of the Uploader on a local file system, please see the [#important Notes for Developers](#) below.

For more details on the Uploader design and design Process, see [Uploader Design Overview](#).

## Progressive Enhancement

The Uploader utilizes the concept of *progressive enhancement*. The goal is to ensure that the page is usable by the widest possible audience, even with old browsers or when JavaScript is turned off. This is done by specifying a regular file input element in the markup. When the Uploader is initialized, the Upload code will replace that element with the Fluid Uploader. As of Infusion 1.3, progressive enhancement will occur automatically by default. It can be overridden by choosing a specific upload strategy instead of using the `fluid.uploader.progressiveStrategy`.

## Upload Strategy

The Infusion Uploader, like many Fluid components, is really one interface to a collection of components that work together to provide a unified user experience.

The Uploader provides a *facade* object, called a *strategy*, which represents the entire subsystem for a particular implementation of Uploader. There are currently two different strategies available to choose from:

1. `fluid.uploader.html5Strategy`, which provides the modern HTML 5 implementation of Uploader
2. `fluid.uploader.swfUploadStrategy`, which implements the legacy implementation of Uploader with Flash and [SWFUpload](#)

To support progressive enhancement, there is a third type of strategy, `fluid.uploader.progressiveStrategy`, which uses the new Infusion [IoC - Inversion of Control System](#) to deliver the best possible version of Uploader based on the capabilities of the user's browser.

The default strategy for Uploader is `fluid.uploader.progressiveStrategy`.

## Upgrading

**Upgrading from Infusion 1.2:** The Uploader was substantially refactored for the Infusion 1.3 in order to support the new HTML 5 version. However, most users should be unaffected. All events, selectors, and classes remain compatible with previous versions. Since the Uploader's underlying structure has changed significantly, and support for Infusion's IoC System was introduced, several other configuration options have changed.

In order to ease the transition, we've provided a compatibility file that will automatically transform your options from the old format to the new when you invoke `fluid.uploader()`. This can be enabled simply by including the `UploaderCompatibility-Infusion1.2.js` file your page. If you're not using a custom build of Infusion, you will also need to include the framework's `ModelTransformations.js` file.

Upgrading from Infusion 0.8: The classnames used for selectors and styles in the Uploader were extensively refactored for Infusion 1.0. Please refer to the [Tutorial - Uploader 1.0 Migration](#) documentation before upgrading from Infusion 0.8.

Upgrading from Infusion 0.5: During Infusion 0.6, the Uploader API had a complete overhaul. Please refer to this document and the latest example code before upgrading from Infusion 0.5 or earlier.

#### Status

This component is in [Preview status](#)

#### On This Page

- [Uploader Overview](#)
  - [Progressive Enhancement](#)
  - [Upload Strategy](#)
  - [Upgrading](#)
- [Creating an Uploader](#)
  - [Parameters](#)
    - [container](#)
    - [options](#)
- [Supported Events](#)
  - [File Objects](#)
  - [File Status Constants](#)
- [Options](#)
  - [Uploader Subcomponents:](#)
  - [Uploader Options:](#)
  - [queueSettings Options](#)
  - [htmlStrategy Options](#)
  - [swfUploadStrategy Options](#)
- [Selectors](#)
- [Dependencies](#)
  - [Required](#)
- [Important Notes for Developers](#)
  - [Running locally: "demo" mode](#)

#### See Also

- [Uploader](#)
- [Uploader Design Overview](#)
- [Tutorial - Uploader](#)
- [Uploader Wireframes \(Design Iteration\)](#)
- [Upload Design Pattern](#)
- [Tutorial - Uploader 1.0 Migration](#)

#### Still need help?

Join the [infusion-users mailing list](#) and ask your questions there.

## Creating an Uploader

To instantiate a new Uploader on your page using the recommended progressive enhancement feature:

```
var myUploader = fluid.uploader(container, options);
```

Returns: A Uploader component object. The resulting type may be either `fluid.uploader.multiFileUploader` or `fluid.uploader.singleFileUploader` depending on the capabilities of your user's browser. If you're programmatically calling methods on the Uploader, be sure to check its `typeName` or use duck typing first.

## Parameters

### container

The `container` is a CSS-based [selector](#), single-element jQuery object, or DOM element that identifies the root DOM node of the Uploader markup.

### options

The `options` object is an optional data structure that configures the Uploader, as described in the [fluid:Options](#) section below.

## Supported Events

The Uploader fires the following events (for more information about events in the Fluid Framework, see [Events for Component Users](#)):

Event	Type	Description	Parameters	Parameter Description
afterReady	default	This event is fired when the Flash (swf) object is instantiated and ready.	none	
onFileDialog	default	The event is fired when the OS File Open dialog is displayed.	none	
onFilesSelected	default	The event is fired when files have been selected for addition to the file queue. HTML5 only.	numFiles	The number of files selected.
onFileQueued	default	The event is fired when files have been added to the file queue. Flash only.	none	
afterFileQueued	default	This event is fired for each file that is queued after the File Selection Dialog window is closed.	file	file: the <a href="#">fluid:File</a> object that has just been added to the queue.
afterFileRemoved	default	Event is fired for each file that is removed from the filequeue model	file	file: the <a href="#">fluid:File</a> object that was just removed from the queue
onQueueError	default	This event is fired for each file that was not queued after the File Selection Dialog window is closed.	file, errorCode	file: the <a href="#">fluid:File</a> object that failed to be added to the queue.  errorCode [fluid:number]: indication of the reason for the file not being added to the queue. Use that.swfUploader.QUEUE_ERROR to interpret the value.  Note: A file may not be queued for several reasons such as, the file exceeds the file size, the file is empty or a file or queue limit has been exceeded.
afterFileDialog	default	This event fires after the File Selection Dialog window has been closed and all the selected files have been processed.	none	
onUploadStart	default	This event at the beginning of the entire upload cycle.	fileList	fileList: a list of the <a href="#">fluid:File</a> objects that will be uploaded to the server.
onFileStart	default	This event is fired immediately before the file is uploaded.	file	file: the <a href="#">fluid:File</a> object that has started to be uploaded.
onFileProgress	default	This event is fired periodically by the upload manager during a file upload and is useful for providing UI updates on the page.  Also keeps up-to-date the overall progress of the currently uploading batch.	file, fileBytesComplete, fileTotalBytes	file: the <a href="#">fluid:File</a> Object for this progress event  fileBytesComplete [fluid:number]: the number of bytes for the current file that have been uploaded to the server  fileTotalBytes [fluid:number]: the total size of the file in bytes.
onFileError	default	This event is fired any time an individual file upload is interrupted or does not complete successfully.	file, error, status, xhr	file: the <a href="#">fluid:File</a> object that errored.  error [fluid:number]: whether the file failed because the user cancelled the upload or because of another file specific error Use fluid.uploader.fileStatusConstants to interpret the value.  status [fluid:string]: xhr.status. 0 if the error occurs during the upload, otherwise, the HTTP status code.  xhr: Defaults to the XMLHttpRequest when available (IE), the XMLHttpRequest otherwise.

onFileSuccess	default	This event is fired when an upload completes and the server returns a HTTP 200 status code.	file, responseText, xhr	file: the <a href="#">fluid:File object</a> that has completed. responseText: the response text returned from the server. xhr: Defaults to the XMLHttpRequest when available (IE), the XMLHttpRequest otherwise.
afterFileComplete	default	This event is fired for each file uploaded whether or not the file is uploaded successfully (onFileSuccess) or the file failed to upload (onFileError). At this point the upload process for the currently uploading file is complete and the upload for the next file in the queue is started.	file	file: the <a href="#">fluid:File object</a> that was successfully uploaded.
afterUploadComplete	default	This event is fired at the end of an upload cycle when all the files have either been uploaded, failed to upload, the user stopped the upload cycle, or there was an unrecoverable error in the upload process and the upload cycle was stopped.	fileList	fileList: the list of <a href="#">fluid:File objects</a> that "completed" (either succeeded or failed), in this upload.

## File Objects

Many of the Uploader's events pass a `File` object as a parameter to the event listener. These objects provide useful information about the file, including its name, size in bytes, and its current status in the queue.

`File` object: a representation of each file in the file queue, as provided from the upload strategy. At the moment, the properties of this object will be slightly different depending on the strategy you're using. This will be addressed in a future release.

Regardless of the strategy, the following properties will be available:

```
id : string,           // a unique id for each file in the queue
name : string,        // The file name. The path is not provided.
size : number,        // The file size in bytes
filestatus : number   // The file's current status.
                      // Use fluid.uploader.fileStatusConstants to interpret the value.
```

## File Status Constants

The Uploader offers a set of constants used to denote the status of a particular file in the queue. These can be used when querying the `filestatus` property of a [fluid:File object](#).

Upload Error Constants	Description
<code>fluid.uploader.errorConstants.HTTP_ERROR</code>	An HTTP error occurred while uploading a file.
<code>fluid.uploader.errorConstants.MISSING_UPLOAD_URL</code>	The <code>uploadURL</code> was not correctly specified in the <code>uploadManager</code> options.
<code>fluid.uploader.errorConstants.IO_ERROR</code>	An IO error occurred while transferring the file.
<code>fluid.uploader.errorConstants.SECURITY_ERROR</code>	The upload caused a security error to occur.
<code>fluid.uploader.errorConstants.UPLOAD_LIMIT_EXCEEDED</code>	The user attempted to upload more files than allowed by the <code>fileUploadLimit</code> option for the <code>uploadManager</code> .
<code>fluid.uploader.errorConstants.UPLOAD_FAILED</code>	The Uploader was unable to start uploading the file to the server
<code>fluid.uploader.errorConstants.SPECIFIED_FILE_ID_NOT_FOUND</code>	This indicates an error in the Uploader and should be filed as a <a href="#">bug</a>
<code>fluid.uploader.errorConstants.FILE_VALIDATION_FAILED</code>	
<code>fluid.uploader.errorConstants.FILE_CANCELLED</code>	The upload was canceled by the user.
<code>fluid.uploader.errorConstants.UPLOAD_STOPPED</code>	The upload was stopped by the user.
File Status Constants	
<code>fluid.uploader.fileStatusConstants.QUEUED</code>	The file is currently queued up and ready to be sent to the server.
<code>fluid.uploader.fileStatusConstants.IN_PROGRESS</code>	The file is currently being uploaded to the server.
<code>fluid.uploader.fileStatusConstants.ERROR</code>	An error occurred while trying to upload the file.
<code>fluid.uploader.fileStatusConstants.COMPLETE</code>	The file was successfully uploaded to the server.
<code>fluid.uploader.fileStatusConstants.CANCELLED</code>	The file was canceled by the user while in the process of being uploaded.

## Options

The Uploader supports a "plug-and-play" architecture that allows for many of the sub-components of the Uploader to be swapped out for other components or your own custom components. The best example of this is the `strategy` component, which allows you to choose between the `fluid.uploader.html5Strategy`, the `fluid.uploader.swfUploadStrategy`, and the `fluid.uploader.progressiveStrategy`. However you can also replace the Progress subcomponent and the FileQueueView subcomponent, with a customized version you have implemented yourself.

The Uploader and its sub-components are also highly configurable; you can make many adjustments to the user experience through a combination of HTML, CSS and the built-in configuration options. To customize the component for your own needs, start with these out-of-the-box features. If you need more flexibility, feel free to write your own sub-component.

In addition to the Uploader options, there are also options specifically for the FileQueueView, Progress, and strategy subcomponents.

## Uploader Subcomponents:

Name	Description	Values	Default
strategy	The strategy for how files are uploaded to the server (e.g. HTML 5, Flash, etc.)	"fluid.uploader.progressiveStrategy", "fluid.uploader.html5Strategy", or "fluid.uploader.swfUploadStrategy"	<pre>strategy: {   type: "fluid.uploader.   progressiveStrategy" }</pre>
fileQueueView	Specifies the type of fileQueueView subcomponent to use. Currently there is only one fileQueueView sub-component.	"fluid.fileQueueView"	<pre>fileQueueView: {   type: "fluid.fileQueueView" }</pre>
totalProgressBar	Specifies the type and options to use for the total progress bar.	See the <a href="#">Progress API documentation</a> for a full descriptions of the available options.	<pre>totalProgressBar: {   type: "fluid.progress",   options: {     selectors: {       progressBar: ".flc-uploader-       queue-footer", (was ".flc-uploader-       scroller-footer" in v1.0)       displayElement: ".flc-uploader-       total-progress",       label: ".flc-uploader-total-       progress-text",       indicator: ".flc-uploader-       total-progress",       ariaElement: ".flc-uploader-       total-progress"     }   } },</pre>

## Uploader Options:

Name	Description	Values	Default
selectors	Javascript object containing selectors for various fragments of the uploader markup		see <a href="#">#Selectors</a> below
listeners	JavaScript object containing events and the listeners that are attached to them.	Keys in the object are event names, values are functions or arrays of functions.	See <a href="#">fluid:Supported Events</a>
focusWithEvent	Javascript object containing selectors for markup elements that should obtain focus after certain <a href="#">#events</a> .	Keys in the object are supported event names. <b>Note:</b> only specific methods in the Uploader have been factored to use this values.	<pre>focusWithEvent: {   afterFileDialog: "uploadButton",   afterUploadStart: "pauseButton",   afterUploadStop: "uploadButton" },</pre>

styles	Specific class names used to achieve the look and feel of the different states of the Uploader		<pre> styles: {   disabled: "fl-uploader-disabled",   hidden: "fl-uploader-hidden",   dim: "fl-uploader-dim",   totalProgress: "fl-uploader-total-progress-okay",   totalProgressError: "fl-uploader-total-progress-errored" } </pre>
strings	Strings that are used in the Uploader. Note: The strings that contain tokens (example: %curFileN) are passed through a string template parser.		<pre> strings: {   progress: {     fileUploadLimitLabel: "%fileUploadLimit %fileLabel maximum",     toUploadLabel: "To upload: %fileCount %fileLabel (%totalBytes)",     totalProgressLabel: "Uploading: %curFileN of %totalFilesN %fileLabel (%currBytes of %totalBytes)",     completedLabel: "Uploaded: %curFileN of %totalFilesN %fileLabel (%totalCurrBytes)%errorString",     numberOfErrors: ", %errorsN %errorLabel",     singleFile: "file",     pluralFiles: "files",     singleError: "error",     pluralErrors: "errors"   },   buttons: {     browse: "Browse Files",     addMore: "Add More",     stopUpload: "Stop Upload",     cancelRemaning: "Cancel remaining Uploads",     resumeUpload: "Resume Upload"   },   queue: {     emptyQueue: "File list: No files waiting to be uploaded.",     queueSummary: "File list: %totalUploaded files uploaded, %totalInUploadQueue file waiting to be uploaded."   } } </pre>
demo	Boolean indicating whether to run in "demo" mode. See <a href="#">Running locally: 'demo' mode</a> below.	false	demo: false

## queueSettings Options

Name	Description	Values	Default
uploadURL	The URL to which files should be sent via POST requests.	String	""
fileUploadLimit	The maximum number of files allowed to be uploaded. 0 is unlimited.	Integer	0
fileQueueLimit	The number of files that can be queued at once before uploading them. 0 is unlimited	Integer	0
postParams	Parameters to send along with the POST request to the server when uploading files.	Javascript Object	{}
fileSizeLimit	The maximum size of a file to send to the server. Files larger than this will not be added to the queue.	Integer, specified in bytes	"20480"
fileTypes	The type of files that are allowed to be uploaded. Each file type should be specified as *. <i>[fluid:file extension]</i> , separated by semicolons. Example: *.jpg;*.jpeg;*.gif;*.tiff	String	<pre> **.*" in v1.0 *** in v1.1 null in v1.4 </pre>

## htmlStrategy Options

Name	Description	Values	Default
legacyBrowserFileLimit	A special file size limit for older browsers (such as Firefox 3.6), which don't fully support HTML 5 file uploads.	any integer value, specified in megabytes	100

## swfUploadStrategy Options

Name	Description	Values	Default
<b>flashSettings</b>			
flashURL	The URL to the SWFUpload Flash file, required for the Uploader to function.	String	"../flash/swfupload.swf"
flashButtonPeerId	The id of the element which will be replaced by the Flash movie when using Flash 10. Ordinarily, this is configured by the <code>swfUploadSetupDecorator</code> and needn't be specified.	String	""
fileTypesDescription	A human-readable description of the types of files allowed. This is displayed in the Browse Files dialog.	String	null
flashButtonAlwaysVisible	Specifies if a visible, Flash-based <i>Browse Files</i> button should always appear, even in Flash 9	boolean	true in v1.0 false in v1.1
transparentEvenInIE	Specifies whether or not the Uploader should be transparent.	boolean	false in v1.0 true in v1.1
flashButtonImageURL	Path to the <i>Browse Files</i> button image. This image should include each of the button states, stacked on top of each other. Take a look at the default <code>browse.png</code> for an example	String	"../images/uploader/browse.png"
flashButtonHeight	The height of the <i>Browse Files</i> button, in pixels.	Number	22 in v1.0; none in v1.1;
flashButtonWidth	The width of the <i>Browse Files</i> button, in pixels	Number	106 in v1.0 none in v1.1
debug	Shows the SWFUpload debug window when set to <code>true</code> .	boolean	false
styles	CSS class names to use when styling the <i>Browse Files</i> overlay button.	Object	<pre> styles: {   browseButtonOverlay: "fl-browse-button-overlay",   flash9Container: "fl-uploader-flash9-container",   (New in v1.1)   uploaderWrapperFlash10: "fl-uploader-flash10-wrapper" (New in v1.1) } </pre>

## Selectors

One of the options that can be provided to the Uploader is a set of CSS-based selectors identifying where in the DOM different elements can be found. The value for the option is itself a Javascript object containing name/value pairs:

```

selectors: {
  selector1Name: "selector 1 string",
  selector2Name: "selector 2 string",
  ...
}

```

The different parts of the Uploader interface each have their own set of selectors (though all selectors must be provided together in a single object). Each also has a default, as defined below:

### General

Selector name	Description	Default
fileQueue	The container element of the File Queue.	".fl-uploader-queue"
browseButton	The Browse Files button.	".fl-uploader-button-browse"
uploadButton	The Upload button.	".fl-uploader-button-upload"

pauseButton	The Pause button.	".flc-uploader-button-pause"
totalFileProgressbar	The file container for the total progress bar.	".flc-uploader-queue-footer"
totalFileStatusText	The element to write the total progress bar status text into.	".flc-uploader-total-progress-text"
instructions	The element containing the browse files instructions.	".flc-uploader-browse-instructions"

### File Queue

Selector name	Description	Default
fileRows	The files rows in the queue.	".flc-uploader-file"
fileName	The container for the file's name. Scoped within an individual file row.	".flc-uploader-file-name"
fileSize	The container for the file's size. Scoped within an individual file row.	".flc-uploader-file-size"
fileIconBtn	The container for the file row icons. Scoped within an individual file row.	".flc-uploader-file-action"
errorText	The container for file specific error text. Scoped within an individual file row.	".flc-uploader-file-error"
rowTemplate	A template element to clone when creating new rows in the file queue.	".flc-uploader-file-tmpl"
errorInfoRowTemplate	A template element to clone when displaying an error for an individual file.	".flc-uploader-file-error-tmpl"
rowProgressorTemplate	A template element to clone when creating the progress bar for a file row.	".flc-uploader-file-progressor-tmpl"

### Scroller

Selector name	Description	Default
wrapper	A wrapper container around the scrollable element.	".flc-scroller"

### Progress

Note: Please see the [Progress API](#) document for a full description of Fluid Progress. Uploader uses the following selector options for Progress:

```

selectors: {
  progressBar: ".flc-uploader-queue-footer",
  displayElement: ".flc-uploader-total-progress",
  label: ".flc-uploader-total-progress-text",
  indicator: ".flc-uploader-total-progress",
  ariaElement: ".flc-uploader-total-progress"
}

```

Any selectors not provided as an option will revert to the default. Implementers may choose to use the default class names in their markup, or customize the selectors, or a combination of these two approaches.

For example, if your markup uses all of the default selectors, except for the file queue selector and the remove button selector, you would provide the following selectors option:

```

selectors: {
  fileQueue: "#my-file-queue",
  removeButton: "#my-remove-button"
}

```

## Dependencies

### Required

The Uploader dependencies can be met by including in the header of the HTML file



- the minified InflationAll.js file
- the Fluid layout CSS file
- the Uploader CSS file

as shown below:

```
<link rel="stylesheet" type="text/css" href="/framework/fss/css/fss-layout.css" />
<link rel="stylesheet" type="text/css" href="components/uploader/css/Uploader.css" />
<script type="text/javascript" src="framework/core/js/InfusionAll.js"></script>
```

Alternatively, the individual file requirements are:

```
<link rel="stylesheet" type="text/css" href="../../framework/fss/css/fss-reset.css" />
<link rel="stylesheet" type="text/css" href="../../framework/fss/css/fss-layout.css" />
<link rel="stylesheet" type="text/css" href="../../css/Uploader.css" />

<!-- Fluid and jQuery Dependencies -->
<script type="text/javascript" src="../../lib/jquery/core/js/jquery.js"></script>
<script type="text/javascript" src="../../lib/jquery/ui/js/jquery.ui.core.js"></script>
<script type="text/javascript" src="../../framework/core/js/jquery.keyboard-ally.js"></script>
<script type="text/javascript" src="../../lib/swfobject/js/swfobject.js"></script>
<script type="text/javascript" src="../../lib/swfupload/js/swfupload.js"></script>
<script type="text/javascript" src="../../framework/core/js/Fluid.js"></script>
<script type="text/javascript" src="../../framework/core/js/FluidDocument.js"></script>
<script type="text/javascript" src="../../framework/core/js/FluidView.js"></script> <!-- New in
Infusion 1.3 -->
<script type="text/javascript" src="../../framework/core/js/DataBinding.js"></script> <!-- New in
Infusion 1.3 -->
<script type="text/javascript" src="../../framework/core/js/FluidIoC.js"></script> <!-- New in
Infusion 1.3 -->
<script type="text/javascript" src="../../framework/enhancement/js/ProgressiveEnhancement.js"><
/script>

<!-- Uploader dependencies -->
<script type="text/javascript" src="../../js/Uploader.js"></script>
<script type="text/javascript" src="../../js/FileQueue.js"></script>
<script type="text/javascript" src="../../js/Scroller.js"></script>
<script type="text/javascript" src="../../progress/js/Progress.js"></script>
<script type="text/javascript" src="../../js/FileQueueView.js"></script>
<script type="text/javascript" src="../../js/FlashUploaderSupport.js"></script> <!-- New in
Infusion 1.3 -->
<script type="text/javascript" src="../../js/Flash9UploaderSupport.js"></script> <!-- New in
Infusion 1.3 -->
<script type="text/javascript" src="../../js/HTML5UploaderSupport.js"></script> <!-- New in
Infusion 1.3 -->
<script type="text/javascript" src="../../js/DemoUploadManager.js"></script>
```

## Important Notes for Developers

### Running locally: "demo" mode

The Upload component requires a server component to accept the *uploaded* files.

However there are times when you want to run the uploader with out a server: when working on integrating the component with your code, developing or testing the UI, or demonstrating the functionality of the code. For that reason the Uploader has a "demo" mode. In demo mode, the Uploader uses a special version of the uploadManager that pretends to be talking to a server. Most of the code is identical to the server mode because the same events are being fired and the model is exactly the same. Most of the code in the Uploader *thinks* that there is a server.

To run locally you must specify `demo: true` in your component configuration:

```
var myUploader = fluid.progressiveEnhanceableUploader(".flc-uploader", ".fl-ProgEnhance-basic", {
  demo: true,
  uploadManager: "fluid.swfUploadManager"
});
```

To run locally, it is also necessary to make some minor modifications to your Flash settings in order to allow the Flash component to view files on your local machine from a locally instantiated Flash file. See [Enabling Uploader on Local File Systems](#) for instructions.