File API <https://drupal.org/node/555118>

File API overview

The Drupal 7 File API provides an abstraction layer for accessing virtually any type of resource as a normal file. This is accomplished by leveraging the power of PHP stream wrappers allowing greater collaboration and crossover with a variety of modules. Modules will typically no longer need to know about the filesystem implementation of another in order to interact.

### Streamwrapper notation for all files

PHP provides the [Streams](http://php.net/manual/en/book.stream.php) capability, which allows all files and devices to be accessed with notation like **public://something.txt** instead of having to figure out where the files directory is, then access it with "sites/default/files/something.txt". **Drupal has extended streams to provide the public files directory (by default at sites/default/files) with public:// notation, the private directory (by default sites/default/private) with private:// notation, and temporary files with temporary://somefile.txt.** Those three schemes (public, private, and temporary) will be used most often to access files. In general, normal paths will not be used at all when using the Drupal File API.

Since PHP provides streams, though, all the PHP streamwrapper notations can be used by the PHP File functions, including plain paths, ftp://, http://, etc.

Drupal also provides an explicit way for a module to add an additional "scheme" by implementing a hook and a class. The File Example in the [Examples Project](http://drupal.org/project/examples) has a [full example implementation](http://api.drupal.org/api/examples/file_example--file_example_session_streams.inc/7) of a session:// scheme that writes "files" to the $\_SESSION variable.

# File usage functions

One of the things to think about when using the Drupal 7 File API is the new file\_usage\_\* functions. These provide a way for modules to register that they are using a file so file\_delete() won't delete the file if it is still required. There are 3 file\_usage\_\* functions:

* [**file\_usage\_list()**](http://api.drupal.org/api/drupal/includes--file.inc/function/file_usage_list/7)**: Lists the usage of a file given the file object as an argument.**
* [**file\_usage\_delete()**](http://api.drupal.org/api/drupal/includes--file.inc/function/file_usage_delete/7)**: Removes a record of a file being used.**
* [**file\_usage\_add()**](http://api.drupal.org/api/drupal/includes--file.inc/function/file_usage_add/7)**: Records that a module uses a file, what object it's used in, what type that object is and the module that is responsible.**

When a file is saved in Drupal 7 it will not automatically register it with file\_usage\_add(). This will have to be done separately. To remove the file you have to use file\_usage\_delete() to remove the listed usage of the file. But remember that this might break other modules, so the **best approach is not to remove the file as long as it's used by another module.**

# General concepts

The Drupal File API leverages [PHP stream wrappers](https://php.net/manual/intro.stream.php) allowing virtually any type of resource to be represented as a normal file. This is a fundamental shift in the way files are managed in comparison to previous versions of Drupal.

#### What are stream wrappers?

Quite simply, **a stream is simple data.** This input could be from a normal file, from standard input (STDIN), or even a Drupal node. A wrapper encompasses these streams and determines how to handle a specific protocol. Let's look at a few examples.

http:// This wrapper is included with PHP and allows you to access a URL on the web such as this page.  
youtube:// This wrapper is implemented by [a Drupal contrib module](http://drupal.org/project/media_youtube) and tells PHP how to handle a YouTube video.

The handling of these streams is done behind the scenes. Developers building on top of registered stream wrappers don't need to know how to use the YouTube API, or how to implement the HTTP protocol. Those specific details are handled by the wrapper. Of course, the wrapper developer needs to know these protocols, but if you're building on top of an existing wrapper, then you don't need to worry about them!

#### "Calling" streams

**A stream is referenced as scheme://target**. The scheme represents the protocol or encoding to be used. For example 'youtube', 'http', 'public', and 'private' are schemes. The target represents the resource desired. This could be a file on the filesystem (public://pictures/me.jpg) or something completely different ([youtube://NIfl2o44zb0](http://www.youtube.com/watch?v=NIfl2o44zb0)). It's that simple! As previously noted, the handling of these resources is performed by the wrapper behind the scenes.

#### Using managed files

**The stream reference is stored in the 'uri' property of a managed file object. Use this reference as a file name for most PHP functions. So, given a file ID number $file\_id, the following code uses [file\_get\_contents()](https://php.net/manual/function.file-get-contents.php) to read the contents of a managed file into a string:**

**$file = file\_load($file\_id);**

**$contents = file\_get\_contents($file->uri);**

#### Limitations

Once the wrappers are registered you can make normal system calls and PHP will transparently handle them. However, there are just a few shortcomings with PHPs wrapper support, but we've replaced the functionality in Drupal's File API.

# Important Drupal 7 File API Functions

All the File API functions are at <http://api.drupal.org/api/drupal/includes--file.inc/7>, but it can be hard to sort out which ones you really want to use. Here's a summary:

* [**file\_save\_data()**](http://api.drupal.org/api/function/file_save_data/7): Writes a buffer to a *managed* file. The file will be named in streams format, as in public://somefile.txt or private://somefile.txt. This **returns a file object** which can be used in relation to other Drupal APIs. Example: $file\_object = file\_save\_data("This data should go in a file", "public://somefile.txt");
* [**file\_unmanaged\_save\_data()**](http://api.drupal.org/api/function/file_unmanaged_save_data/7): Writes a file just like file\_save\_data(), but nothing is done to the database; the file is not a Drupal object in any way, it's just a file. This function **returns the filepath**. Example: $filepath = file\_unmanaged\_save\_data("This data should go in a file", "public://somefile.txt");
* [**file\_copy()**](http://api.drupal.org/api/function/file_copy/7) and **[file\_move()](http://api.drupal.org/api/function/file_move/7)**: **Take a file object and copy or move to a stream path, returning a file object**. Example: $file\_object = file\_copy($some\_file\_object, "public://somefile.txt");
* [**file\_unmanaged\_copy()**](http://api.drupal.org/api/function/file_unmanaged_copy/7) and **[file\_unmanaged\_move()](http://api.drupal.org/api/function/file_unmanaged_move/7)**: **Take a stream filepath and copies/moves to another stream filepath, returning a filepath**. Example: $filepath = file\_unmanaged\_copy("private://afile.txt", "public://somefile.txt");
* [**file\_prepare\_directory()**](http://api.drupal.org/api/function/file_prepare_directory/7): **Checks, creates, sets permissions on a directory path.** Example: $directory = "public://somedir/anotherdir"; $result = file\_prepare\_directory($directory, FILE\_MODIFY\_PERMISSIONS | FILE\_CREATE\_DIRECTORY);. This will attempt to create the directory and set standard permissions on it, returning FALSE if it fails. The $directory must be passed as a variable.
* [**file\_delete()**](http://api.drupal.org/api/function/file_delete/7): **Delete a file and its associated record in the file\_managed table**. May return TRUE for success, FALSE for failure, or an array of references to the file that prevented it from being deleted. Example: $result = file\_delete($file\_object, TRUE);. (Because this example sets the **second parameter $force = TRUE, file\_delete will attempt to delete the physical file whether or not other modules had references to it.**).
* [**file\_unmanaged\_delete()**](http://api.drupal.org/api/function/file_unmanaged_delete/7) and **[file\_unmanaged\_delete\_recursive()](http://api.drupal.org/api/function/file_unmanaged_delete_recursive/7)**: **Delete a file (or directory tree) from the filesystem**. Returns TRUE for success. Examples: $result = file\_unmanaged\_delete("public://somefile.txt");or $result = file\_unmanaged\_delete\_recursive("public://somedir");
* [**file\_create\_url()**](http://api.drupal.org/api/function/file_create_url/7): **Translate a file path like public://somefile.txt into an accessible URL** like <http://example.com/sites/default/files/somefile.txt>. Example: $url = file\_create\_url("public://somefile.txt");

### PHP File Functions

The Drupal File API is intended to be the default way to work with files in Drupal, especially files that need to be managed in the database. However the Drupal functions are not intended to completely replace the PHP file functions, so don't forget about [PHP functions](http://php.net/manual/en/function.file.php) including **file(), to read a file into an array, file\_get\_contents()/file\_put\_contents() to read/write from a buffer, fopen()/fread()/fwrite(), is\_dir(), is\_file((), is\_readable(), is\_writable(),** etc. In many simple cases the Drupal API will provide what's necessary, but the vast majority of the PHP functions work perfectly with streams, so will work with Drupal schemes like public://somefile.txt and private://anotherfile.txt as well as PHP-provided schemes like <https://example.com/somefile.html> and <ftp://ftp@example.com/some/file.txt>.

### Example Code

The File Example in the [Examples Project](http://drupal.org/project/examples) has a [full example implementation](http://api.drupal.org/api/examples/file_example--file_example_session_streams.inc/7) of a session:// scheme that writes "files" to the $\_SESSION variable.

# Writing stream wrappers

This document will first discuss the implementation of a simple local stream wrapper, similar to the public wrapper included with Drupal core. Following that, the Drupal stream wrapper registry will be introduced and you will learn how to register your custom wrapper.

**Sample wrapper (foobar://)**

Example below implements stream for default Drupal distribution files, which should be located separately from public files dir, since it's not existent when distro is installed, but some files should exist by that time anyway (like image fields in sample content of a distro).

/\*\*

\* **Implements hook\_stream\_wrappers().**

\*/

function MYMODULE\_stream\_wrappers() {

return array(

'foobar' => array(

'name' => t('Default distribution files'),

'class' => 'FooBarStreamWrapper',

'description' => t('Provides read-only paths to default distribution files.'),

'type' => STREAM\_WRAPPERS\_READ\_VISIBLE,

),

);

}

/\*\*

\* Default files (foobar://) stream wrapper class.

\*/

class FooBarStreamWrapper extends DrupalPublicStreamWrapper {

public function getDirectoryPath() {

return 'profiles/mydistro/files';

}

}

After this you could just replace public:// paths in database with foobar:// paths.

**Stream wrapper registry**

**The stream wrapper registry in Drupal keeps track of the scheme and implementation class of each registered wrapper. It is important to understand that the Drupal stream wrapper registry differs from PHP's native registry. The Drupal registry builds on top of PHPs native support and adds more functionality**. For example, PHP does not provide native capabilities to ask "what wrapper class is responsible for handling this URI or scheme?" This ability is needed in order to provide functions specific to Drupal such as drupal\_chmod().

**The wrapper registry does not register PHP's built-in wrappers such as http or ssl.** These wrappers may still be used with PHP functions normally, but any functions that rely on the Drupal registry will fail because the wrapper will not be found. For example, the call file\_stream\_wrapper\_valid\_scheme('http') would return false despite PHP's built-in http wrapper being available. In most cases this will not be an issue because PHP's wrappers do not contain attributes specific to Drupal.

**Registering wrappers**

This is a pretty straightforward procedure involving hook\_stream\_wrappers() which returns an array of wrappers. Each wrapper having a name, description, type and class. Class being the actual PHP class of the wrapper and type being a constant with these possible values:

* STREAM\_WRAPPERS\_HIDDEN: not visible in the UI or accessible via web, but readable and writable. E.g. the temporary directory for uploads.
* STREAM\_WRAPPERS\_LOCAL\_HIDDEN: hidden, readable and writeable using local files.
* STREAM\_WRAPPERS\_LOCAL\_NORMAL: visible, readable and writeable using local files.
* STREAM\_WRAPPERS\_NORMAL: The default when type is omitted, do not include the STREAM\_WRAPPERS\_LOCAL flag. Read more about it here
* STREAM\_WRAPPERS\_READ\_VISIBLE: visible and read-only.
* STREAM\_WRAPPERS\_WRITE\_VISIBLE: visible, readable and writeable.

You can read more about all of these at the stream\_wrappers.inc api page.

function flickrview\_stream\_wrappers() {

return array(

'flickr' => array(

'name' => t('Flickr'),

'class' => 'FlickrStreamWrapper',

'description' => t('Stream wrapper for reading files from flickr'),

'type' => STREAM\_WRAPPERS\_READ\_VISIBLE,

),

);

}

# File API changes from Drupal 6 to Drupal 7

There are three key and pervasive things that happened to the File API between Drupal 6 and Drupal 7:

* **Any API that used to take a traditional filepath like "sites/default/files/something.txt" now must take a stream-oriented file path like "public://something.txt" or "private://something.txt".**
* **Many APIs used to take a string if they were acting on an unmanaged file, or a file object if acting on a managed file**. Now those functions have been split out, so we now have file\_copy() (which takes a file object) and file\_unmanaged\_copy() (which takes a stream-oriented filepath like "public://example.txt").
* **file\_create\_path() and file\_check\_directory() were merged into file\_prepare\_directory().**
* A smaller item: **drupal\_realpath() must be used in place of realpath().**

#### Summary of managed/unmanaged File API changes

|  |  |  |
| --- | --- | --- |
| **Drupal 6** | **Drupal 7** | **Description** |
| file\_copy() | file\_unmanaged\_copy() | Copy a file to a new location without saving a record in the database. |
| n/a | file\_copy() | Copies a file to a new location and adds a file record to the database. Also invokes hook\_file\_copy() so that other modules may act on the copy action. |
| file\_move() | file\_unmanaged\_move() | Move a file to a new location but make no changes to the database. |
| n/a | file\_move() | Move a file to a new location and update the file's database entry. Also invokes hook\_file\_move() so that other modules may act on the move action. |
| file\_delete() | file\_unmanaged\_delete() | Delete a file. |
| n/a | file\_delete() | Delete a file and its database record. Also invokes hook\_file\_delete() to let other modules perform clean-up actions when file is deleted. |
| file\_save\_data() | file\_unmanaged\_save\_data() | Save a string to the specified destination but makes no changes to the database. |
| n/a | file\_save\_data() | Save a string to the specified destination and create a database file entry. |
| n/a | file\_load() | Load a file object from the database. Also invokes hook\_file\_load() to allow other modules to do things as the file is loaded. |
| n/a | file\_validate() | Check that a file meets the criteria specified by the validators. Accepts an associative array of callback functions used to validate the file. Also calls hook\_file\_validate() to let other modules perform validation on the new file. |
| n/a | file\_save() | Save a file object to the database. Calls either hook\_file\_insert() or hook\_file\_update(), depending on whether a $file->fid is specified. |