

PHP HACKS™

*Tips & Tools for Creating
Dynamic Web Sites*



O'REILLY®

Jack D. Herrington

HACK
#33

Access Your iPhoto Pictures with PHP

Use PHP's XML capabilities to parse through iPhoto's picture database.

Apple is a company known for producing innovative and easy-to-use products. Following on that line, it recently released the iLife suite (<http://www.apple.com/ilife/>), which makes it easy to produce and organize rich media. I was a bit dismayed by my options for sharing my photos from iPhoto, though. In particular, after having imported my digital photos from my camera and organizing them using iPhoto, I wanted to show off these pictures to family and friends. I didn't want to sign up for hosting, open an account with a photo printing service, wait for hundreds of files to upload somewhere, export photos to a smaller size, or reorganize all of my images in some other program after having already done the work in iPhoto. I wanted them available to everybody—right now—and I didn't want to have to lift a finger to make it so. I'd already done plenty of work by taking the actual photos, not to mention organizing and captioning them!

This is what got me working on myPhoto (<http://agent0068.dyndns.org/~mike/projects/myPhoto>). One Mac OS X feature that most users often do not notice is the built-in web server; Mac OS X includes both Apache and PHP, and both are itching to be enabled. When you combine this and a broadband connection with all of the information readily available in iPhoto, sharing photos becomes (as it should be) a snap.

If your PHP project requires a photo gallery component, it might be tempting to place the burden on users to upload, caption, and organize all of their photos into your system. However, if users have already done the work in iPhoto, do the rest for them! Armed with a simple XML parser, it's possible to extract all of the meaningful data from iPhoto and reformat it into a simpler format that's more appropriate and convenient for use with PHP.

A Look Behind the Scenes: iPhoto Data

The first logical step is to get up close and personal with iPhoto so that you know what data is easily available.



I am basing this discussion on iPhoto Version 5.x, the most current version of iPhoto available as of this writing. With a few small tweaks here or there, though, it's trivial to apply these same concepts to other versions of iPhoto—something I've been doing since iPhoto 2.0.

Figure 4-14 shows a small selection from my iPhoto album.



Figure 4-14. iPhoto showing pictures from my wedding

A quick look in `~/Pictures/iPhoto Library/` shows almost everything we could ever need from iPhoto:

Directories broken down by date

For instance, `~/Pictures/iPhoto Library/2005/07/02/` contains photos from July 2, 2005. The image files in this directory are the actual full-size photos, but they contain all of the edits the user made from within iPhoto (i.e., rotations, color corrections, etc.). It also contains two other subdirectories: *Thumbs*, which contains 240×180 thumbnails corresponding to each image, and *Originals*, which contains the original, unmodified versions of the images (only if the user has performed any edits in iPhoto). Furthermore, in nearly all cases, these photos are in JPEG format, which is perfect for the Web.



One notable exception: if the user takes photos in RAW format (available on higher-end cameras), the *Originals* directory contains the RAW files and all other images are JPEG representations.

AlbumData.xml

This XML document contains all of the really interesting (and uninteresting) data surrounding these photos: file paths for a given photo, captions, ratings, modification dates, etc. This file also contains information about groups of photos—also called *albums*—as well as user-defined keywords. Some version information and meta-information is included as well, but that’s not terribly helpful.

So now we need to make some sense of that *AlbumData.xml* file. First off, it’s not just any XML file; it’s an Apple Property List. This means that a limited set of XML tags is being used to represent common programmatic data structures like strings, integers, arrays, and dictionaries (also known as *associative arrays* in some languages). Therefore, for the interesting structures within this file, we should look at some sample content, since the XML tags themselves aren’t terribly descriptive. Rather, the tagged content is where the meaty structure is. I’ve cut some pieces out for the sake of brevity, but the more important parts of the file are here.

The beginning of the file looks something like this—not terribly interesting:

```
<?xml version="1.0" encoding="UTF-8"?>
<plist version="1.0">
<dict>
  <key>Application Version</key>
  <string>5.0.4 (263)</string>
  <key>Archive Path</key>
  <string>/Users/mike/Sites/myPhoto/iPhoto Library</string>
```

But further down is a listing of all the photos in the dictionary keyed by unique identifiers for each photo. In the following example, you can see that we’re looking at an individual photo with a unique ID of 5. Furthermore, it’s an image (rather than, say, a video) which has a caption of “No more pictures, please” as well as an optional keyword associated with it (the keyword’s unique keyword ID is 2):

```
<key>Master Image List</key>
<dict>
  <key>5</key>
  <dict>
    <key>MediaType</key>
    <string>Image</string>
    <key>Caption</key>
    <string>No more pictures, please</string>
    <key>Aspect Ratio</key>
    <real>0.750000</real>
    <key>Rating</key>
    <integer>0</integer>
```

```

<key>DateAsTimeInterval</key>
<real>62050875.000000</real>
<key>ImagePath</key>
<string>/Users/mike/Sites/myPhoto/iPhoto Library/2002/12/19/DSC00107.JPG</
string>
<key>OriginalPath</key>
<string>/Users/mike/Sites/myPhoto/iPhoto Library/2002/12/19/Originals/
DSC00107.JPG</string>
<key>ThumbPath</key>
<string>/Users/mike/Sites/myPhoto/iPhoto Library/2002/12/19/Thumbs/5.jpg</
string>
<key>Keywords</key>
<array>
  <string>2</string>
</array>
</dict>
<key>6</key>
...and so on...
</dict>

```

Another section of this file (shown in the next fragment of XML) lists all user-defined groups of photos, known in iPhoto as *albums*. These are stored in a user-defined order in an array (unlike the Master Image List, which is unordered and stored by keys). This includes all kinds of albums—normal albums, smart albums, folders, slideshow albums, book albums, etc. Various album attributes are described—a unique ID, a name, an ordered list of photo IDs for photos contained in the album, an indicator if the album is the “master” album (each photo library should have only one master album), the parent album ID if this album is in a “folder album,” etc.:

```

<key>List of Albums</key>
<array>
<dict>
  <key>AlbumId</key>
  <integer>2</integer>
  <key>AlbumName</key>
  <string>Vacation to somewhere</string>
  <key>KeyList</key>
  <array>
    <string>4425</string>
    <string>4423</string>
    <string>4421</string>
    <string>4419</string>
  </array>
  <key>Master</key>
  <true/>
  <key>PhotoCount</key>
  <integer>2868</integer>
  <key>Parent</key>
  <integer>2196</integer>
</dict>

```

```
<dict>
...and so on...
</dict>
</array>
```

Also worth noting is that there is a structure whose key is “List of Rolls,” which is structurally identical to “List of Albums.” This automatically-generated list groups photos together each time they are imported into iPhoto, treating the group as if it were one “roll” of film.

Finally, the last major section of the file is the list of keywords, a dictionary keyed by IDs. These are user-defined keywords that you can use to tag multiple photos, instead of manually captioning each photo with the same word. This consists of ID/keyword pairs; in this example, the ID is 1 and the keyword is `_Favorite_`:

```
<key>List of Keywords</key>
<dict>
<key>1</key>
<string>_Favorite_</string>
<key>2</key>
<string>...and so on...
</dict>
```



Keep in mind that in older versions of iPhoto, the file format is slightly different; be sure you know and understand this file for the versions of iPhoto you plan on being compatible with. Minor details do change periodically, and they can cripple your parsing code if you don't anticipate or account for them.

The Code

Save the code in [Example 4-10](#) as `iphoto_parse.php`.

Example 4-10. Handling iPhoto XML parsing

```
<?php
//$curTag denotes the current tag that we're looking at in string-stack form
//$curKey denotes the current tagged attribute so that we have some recollection
//of what the last seen attribute was.
// i.e. $curKey="AlbumName" for <key>AlbumName</key>
//$data denotes the element between tags.
// i.e. $data="Library" for <string>Library</string>
//When reading code, note that $curKey is not necessarily equal to $data.

$curTag="";
$curKey="";
$readingAlbums=false;
$firstTimeAlbum=true;
$firstTimeAlbumEntry=true;
```

Example 4-10. Handling iPhoto XML parsing (continued)

```
$readingImages=false;
$firstTimeImage=true;
$firstTimeImageEntry=true;
$curID=0;

$masterImageList=array();

class Photo
{
    var $Caption;
    var $Date;
    var $ImagePath;
    var $ThumbPath;
}

function newPhoto($capt, $dat, $imgPath, $thumb) {
    $aPhoto=new Photo();
    $aPhoto->Caption=$capt;
    $aPhoto->Date=$dat;
    $aPhoto->ImagePath=$imgPath;
    $aPhoto->ThumbPath=$thumb;
    return $aPhoto;
}

//this function is called on opening tags
function startElement($parser, $name, $attrs)
{
    global $curTag;
    $curTag .= "^$name";
}

//this function is called on closing tags
function endElement($parser, $name)
{
    global $curTag;
    $caret_pos = strrpos($curTag, '^');
    $curTag = substr($curTag,0,$caret_pos);
}

//this function has all of the real logic to look at what's between the tags
function characterData($parser, $data)
{
    global $curTag, $curKey, $outputAlbums, $outputImages,
        $readingAlbums, $firstTimeAlbum, $firstTimeAlbumEntry,
        $readingImages, $masterImageList, $firstTimeImage,
        $firstTimeImageEntry, $curID;

    //do some simple cleaning to prevent garbage
    $data = str_replace('!$-a-0*', '&', $data);
    if(!ereg("(\t)+(\n)?$", $data) && !ereg("^\n$", $data))
        //if $data=non-whitespace
```

Example 4-10. Handling iPhoto XML parsing (continued)

```

{
//some common place-signatures...really just a list of unclosed tags
$albumName = "^PLIST^DICT^ARRAY^DICT^KEY"; //album attributes, i.e
    "AlbumName"
$integerData = "^PLIST^DICT^ARRAY^DICT^INTEGER"; //album ID
$stringData = "^PLIST^DICT^ARRAY^DICT^STRING"; //the actual album name
$albumContents = "^PLIST^DICT^ARRAY^DICT^ARRAY^STRING"; //photo ID number
$majorList = "^PLIST^DICT^KEY"; // "List of Albums", "Master Image
    List"
$photoID = "^PLIST^DICT^DICT^KEY"; //the unique ID of an individual
    photo
$photoAttr="^PLIST^DICT^DICT^DICT^KEY"; //"Caption", "Date", "ImagePath", etc
$photoValStr="^PLIST^DICT^DICT^DICT^STRING"; //caption, file paths, etc
$photoValReal="^PLIST^DICT^DICT^DICT^REAL"; // date, aspect ratio, etc

if($curTag == $majorList)
{
    if($data=="List of Albums")
    {
        //flag so that there's no ambiguity, i.e. for <key>List of Rolls</key>
        $readingAlbums=true;
        $readingImages=false;
    }
    else if($data=="Master Image List")
    {
        $readingAlbums=false;
        $readingImages=true;
    }
    else
        $readingAlbums=false;
}

if($readingAlbums)
{
    if ($curTag==$integerData)
    {
        if($data == "AlbumId")
        {
            $curKey = $data;
        }
    }
    else if ($curTag==$albumName) //we're looking at an attribute, i.e
        AlbumName
    {
        //so the next thing we'll see is the album name
        //or the listing of all photos contained in the album
        if($data == "AlbumName" || $data="KeyList")
        {
            $curKey = $data; // $curKey will be that reminder for us next time
        }
    }
}
}

```


Example 4-10. Handling iPhoto XML parsing (continued)

```

    }
    else if($curTag==$photoValStr || $curTag==$photoValReal)
    {
        if($curKey == "Caption" || $curKey == "DateAsTimeInterval" ||
            $curKey=="ImagePath" || $curKey=="ThumbPath")
        {
            if(!$firstTimeImageEntry)
                $curID.=" ";

            if($curKey=="Caption")
                $curID .= "\"caption\"=>\"".addslashes($data).\"";
            else if($curKey=="DateAsTimeInterval") //timeinterval based dates
                                                    //are measured in seconds from 1/1/2001
                $curID .= "\"date\"=>\"".
                    date("F j, Y, g:i a", mktime(0,0,$data,1,1,2001)).
                    "\"";
            else
                $curID .= "\"$curKey\"=>\"$data\"";
            $firstTimeImageEntry=false;
        }
        if($curKey=="ThumbPath") //the last attribute we see for a photo...
            fwrite($outputImages,$curID,'a');
        //...and any other image data worth extracting...
    }
}
}
}

//this function is what you call to actually parse the XML
function parseAlbumXML($albumFile)
{
    global $outputAlbums, $outputImages;
    $xml_parser = xml_parser_create();
    xml_parser_set_option($xml_parser, XML_OPTION_CASE_FOLDING, true);
    //hook the parser up with our helper functions
    xml_set_element_handler($xml_parser, "startElement", "endElement");
    xml_set_character_data_handler($xml_parser, "characterData");
    if (!$fp = fopen($albumFile, "r"))
        die("Can't open file: $albumFile");
    fwrite($outputAlbums,"<?php\n\$albumList = array (\n','w');");
    fwrite($outputImages,"<?php\n//key=photo ID, value={'','w');");
    fwrite($outputImages," [0]caption, [1]date, [2]image ","w');");
    fwrite($outputImages,"path, [3]thumb path}\n\$masterList = array (\n','w');");
    while ($data = fread($fp, 4096))
    {
        $data = str_replace('&', '!$-a-0*', $data);
        if (!xml_parse($xml_parser, $data, feof($fp)))
        {
            die(sprintf("$albumFile : ".$lang["errXMLParse"].": %s at line %d",
                xml_error_string(xml_get_error_code($xml_parser)),
                xml_get_current_line_number($xml_parser)));
        }
    }
}

```

Example 4-10. Handling iPhoto XML parsing (continued)

```

    }
    fwrite($outputAlbums, "\n\t\t)\n\t\n\n);\n?>", 'a');
    fwrite($outputImages, "\n);\n?>", 'a');
    //we're done, throw out the parser
    xml_parser_free($xml_parser);
    echo "Done parsing.";
}

function fwrite($dest, $dataToWrite, $writeMode)
{
    global $err;
    if (is_writable($dest))
    {
        if (!$fp = fopen($dest, $writeMode))
            $err .= "Can't open file: ($dest) <br>";
        else
        {
            if (!fwrite($fp, $dataToWrite))
                $err .= "Can't write file: ($dest) <br>";
            fclose($fp);
        }
    }
    else
        $err .= "Bad file permissions: ($dest) <br>";
}

set_time_limit(0);    //if you have an enormous AlbumData.xml,
//PHP's default 30-second execution time-out is the enemy

$outputImages="out_images.php";
$outputAlbums="out_albums.php";
parseAlbumXML("myPhoto/iPhoto Library/AlbumData.xml");
?>

```

Also, to use the output from the preceding parser, save the code in [Example 4-11](#) as *iphoto_display.php*; this file will handle displaying the photos on the Web.

Example 4-11. The script displaying the photos

```

<?php
include "out_images.php";
$photoIDs=array_keys($masterList);
$thumbsPerPage=6;
$thumbsPerRow=3;
if(!isset($_GET["tStart"]))
    $thumbStart=0;
else
    $thumbStart=$_GET["tStart"];
if($thumbStart+$thumbsPerPage>count($photoIDs))
    $thumbLimit=count($photoIDs);

```

Example 4-11. The script displaying the photos (continued)

```
else
    $thumbLimit=$thumbStart+$thumbsPerPage;
echo "<table border=\"0\" width=\"100%\">\n";
for($x=$thumbStart; $x<$thumbLimit; $x++)
{
    $aPhoto=$masterList[$photoIDs[$x]];
    $thumb="<table>";
    $thumb.="<tr><td align=\"center\"><img ";
    $thumb.="src=\"".$aPhoto["ThumbPath"]."\"></td></tr>";
    $thumb.="<tr><td align=\"center\"><small>";
    $thumb.=$aPhoto["date"]."<br>".$aPhoto["caption"]."</small></td></tr>";
    $thumb.="</table>";
    if($x % $thumbsPerRow == 0)
        echo "\n<!--New row-->\n<tr><td>\n".$thumb."\n</td>\n";
    else if($x % $thumbsPerRow == ($thumbsPerRow-1))
        echo "\n<td>\n".$thumb."\n</td></tr>\n<!--End row-->\n";
    else
        echo "\n<td>\n".$thumb."\n</td>\n";
}
echo "\n</table>\n";
?>
```

Running the Hack

The last few lines of *iphoto_parse.php* contain hardcoded paths to the *AlbumData.xml* file, as well as to the output files (as does *iphoto_display.php*), so be sure that you enter the correct paths. Then, simply load up *iphoto_parse.php* in your web browser. Also, note that PHP will need to have permission to write to the output files; otherwise, you'll get no output.

Your web browser will indicate when the script has finished executing with a page that says, "Done parsing." Open the output files, and you should see an array in each, similar to the following samples.

out_albums.php will look something like this:

```
<?php
$albumList = array (
    "Library" =>
        array(
            4425,
            4423,
            ...
            3796,
            3794,
            3792
        )
)
```

```
);
?>
```

And *out_images.php* will look something like this:

```
<?php
//key=photo ID, value={ [0]caption, [1]date, [2]image path, [3]thumb path}
$masterList = array (
    "13"=>array(
        "caption"=>"The wreath, out of focus again",
        "date"=>"December 23, 2002, 2:59 am",
        "ImagePath"=>"~/mike/myPhoto/iPhoto Library/2002/12/22/DSC00151.JPG",
        "ThumbPath"=>"~/mike/myPhoto/iPhoto Library/2002/12/22/Thumbs/13.jpg"),
    ...
);
?>
```

You can also examine some of the resulting output visually by loading up *iphoto_display.php* in your web browser, as shown in [Figure 4-15](#).



Figure 4-15. iPhoto wedding photos in my browser

While XML is a versatile format, considering how verbose the *AlbumData.xml* file is and how large it can get for photo libraries of even moderate size,

it needs to be massaged. After all, I have only 2,868 photos in my library, but my *AlbumData.xml* file is 2.4 MB. I thus chose to employ the XML parser included with PHP 4 (*expat*) to parse *AlbumData.xml* into meaningful components, which I then output using a much simpler format. Specifically, the output is piped into two separate files containing the data of interest represented as PHP arrays.

The core idea for the parser is to use a string representing the hierarchy of tags so that we have some context as we walk through the file's content. It's sort of like a stack that is represented as a string rather than as the more common array or linked list. Note that this parser parses only some of the elements of the albums section, as well as the images section of *AlbumData.xml*. I've also included a demonstration as to how you can work with the resulting output of this parser.

Before writing any code, it's probably a good idea to decide how to serve your photos. For instance, by default, Mac OS X will not allow Apache (and therefore, PHP) access to *~/Pictures/* where iPhoto data is stored, so you need to get your permissions straight. You can approach this in a number of ways:

- Modify your */etc/httpd/httpd.conf* file.
- Use a symbolic link.
- Quit iPhoto, move your iPhoto *Library* folder into your *~/Sites/* folder, relaunch iPhoto, and when it panics that all the photos are gone, point it to the new location of the *Library* folder.
- Upload your iPhoto *Library* folder to some other machine using FTP, rsync, or any other file-transfer program that floats your boat.

Hacking the Hack

You have a lot of room to work with this hack:

- Add further cases to the XML parser so that it extracts all of the data that you're interested in, rather than just the albums and the images that they contain.
- Instead of outputting the processed *AlbumData.xml* file into a flat text file, store the information in an SQL database or some other, more versatile format.
- If you're going to be this user friendly by getting all of the information out of iPhoto, why not go the extra mile and make this entire process automatic? Automating this process is actually very simple. At this point, we have a means for parsing the XML file as well as a means for

caching what we discover from parsing the XML file. The final step calls for knowing when we should be using the cache and when we should be rebuilding the cache. The answer to this question depends on your application, but here are some possibilities worth considering:

- Run a cron job that invokes your cache rebuild function hourly/daily/whenever.
- Keep track of the modification date of *AlbumData.xml*. If that date is newer than the last time you parsed it, reparse.

So, for example, using the latter approach, add a function that looks something like this:

```
//returns a boolean value indicating whether or not
//a cache rebuild (reparse) is necessary
function needToUpdateCache()
{
    global $cacheTime, $albumFile, $err;

    $cacheTimeFile="lastCacheTime.txt"; //text file where
        //a string indicates
        //last cache rebuild time.
        //i.e. "January 28 2005 16:31:26."
    $compareFile="iPhoto Library/AlbumData.xml";
    if (file_exists($cacheTimeFile))
    {
        //first, check the file where the last known cached time was stored
        if($fp = fopen($cacheTimeFile, "r"))
        {
            $lastTime = fread($fp, filesize($cacheTimeFile));
            fclose($fp);
        }
        else
        {
            $err.= "Can't read last cache time";
            return true;
        }
    }

    //now, determine the last time the iPhoto data has changed
    //if we need to reparse, it will write the
    //current time into $cacheTimeFile
    //(since we will therefore reparse now)
    if($lastTime!=date ("F d Y H:i:s.", filemtime($compareFile)))
    {
        if (!$fp = fopen($cacheTimeFile, 'w'))
        {
            $err.= "Can't open file: $cacheTimeFile";
        }
    }
}
```

```
    }
    else
    {
        if (!fwrite($fp, date ("F d Y H:i:s.", filetype($compareFile)) ))
            $err.= "Can't open file: $cacheTimeFile";
        fclose($fp);
    }
    return true;
}
else
    return false;
}
else
{
    $err.= "Can't find file: $cacheTimeFile";
    return true;
}
}
```

```
//and at the beginning of every page load, call this to ensure
//viewers are getting the latest photos
if(needToUpdateCache())
    parseAlbumXML($pathToYourAlbumXMLFile);
```

This will ensure that you parse the file only when changes have been made in iPhoto that will require a reparse.

—*Michael Mulligan*

See Also

- “Create Thumbnail Images” [Hack #27]
- “Create Image Overlays” [Hack #32]