

Optimizing Your Custom Web Solutions

by Tim Dietrich

If you're developing Web applications that use the FileMaker API for PHP, it's highly likely that you've had situations where an applications isn't quite as fast as you'd like it to be. We all run into this occasionally, and it can be very frustrating. It's particularly frustrating when code that originally ran very quickly now seems to run slowly, and even more so when your users start complaining.

In this article, I'll give you two of the techniques that I use to optimize Web applications built using the FileMaker API for PHP. While there are several other techniques that I use as well, these two techniques typically provide the biggest impact on performance. And best of all, they're both easy to implement.

Streamline Your Layouts

As you build Web solutions that use the FileMaker API for PHP, it can be tempting to use the same layouts that you've already developed for the database's FileMaker Pro users. It's highly likely that there are existing layouts that have the fields that you need for the Web application, so it seems logical to use those layouts for the Web app. There's no harm in using those existing layouts... Right?

Think again! The layouts that you reference via the API do have a very significant impact on your Web application, so you need to be very careful in selecting them. Here's why: When FileMaker receives a request for data from the API, it returns all of the data for every field on that the layout. It also returns all related data that exists in the layout's portals. And to make matters worse, there's a significant amount of meta data ("data about the data") that FileMaker returns as well. So if you've got a layout with a dozen fields on it, and your Web application only really uses three of those fields, FileMaker is sending back a lot more data than is necessary.

Suppose that we're building a Web application that queries a product database for a list of product categories. For example:

```
// Get the category data from the database.
$find_request = $fm->newFindCommand("Category Editor");
$find_request->addFindCriterion("Is_Online", "Yes");
$find_request->addSortRule('Category_Name', 1);
$find_result = $find_request->execute();
if (FileMaker::isError($find_result)) {
```

```
        die ("Database Error");
    }
    $categories = $find_result->getRecords();
```

In this example, we're using a layout named "Category Editor" -- the same layout used by the database's users to maintain product categories. It's a basic layout with 10 fields on it -- the name of the category, a description of the category, the category ID, a field indicating whether the category is active (online) or not, and six fields used for auditing purposes (the name of the user who created the record, the account name of the user that created the record, the date and time that the record was created, the name of the user who last modified the record, the account name of the user that last modified the record, and the date and time that the record was last modified).

With only 8 records in my sample database, the size of the data returned by FileMaker to PHP was a whopping 1.2Mb, and it took about a half a second for PHP to receive it. That's a lot of data. And while that half second might not seem significant, if this code is being called frequently by the Web application, the time adds up very quickly.

There are quite a few fields on the "Category Editor" layout that the Web application simply doesn't need. In this case, it doesn't need any of those six audit fields. So, to optimize the application, we create a new layout designed specifically for the Web application, that contains only the four fields that the application really needs.

The result? The size of the data returned by FileMaker is now reduced from 1.2Mb to 268kb, and it now takes only a quarter of a second for PHP to receive it.

To summarize: Be careful when selecting the layouts that you're going to reference via the API. And if necessary, create streamlined layouts specifically for your Web applications.

Cache Your Data

This is a somewhat advanced technique, but it is fairly easy to implement. And like the layout optimization technique mentioned earlier, it can have a very significant impact on your application's performance. The technique involves storing the data that we've retrieved from the FileMaker database, so that we don't need to make repeated requests for the data.

Continuing with the product category example above, let's suppose that our Web application displays a list of categories on every page. That means that every time a page is requested, we're going back to the FileMaker database and asking it for the list

of categories again. This is very inefficient, because the categories don't change very often.

The solution? Caching the data. We want to ask FileMaker for the category data only once, and store ("cache") the data so that we can refer to it over and over again.

To implement this technique, we're going to use PHP sessions. With sessions, we can store information on the server for each user, and refer to that information repeatedly as the user moves from page to page within the application. Sessions are temporary. They typically exist only during the time that the user is interacting with Web application. They usually expire when the user leaves the site or closes their browser.

Here's a code snippet showing how this technique works:

```
// Create a session or resume the current one.
@session_start();

// If we haven't already cached the categories...
if (!isset($_SESSION['categories'])) {

    // Get the category data from the database.
    $find_request = $fm->newFindCommand("Category Editor");
    $find_request->addFindCriterion("Is_Online", "Yes");
    $find_request->addSortRule('Category_Name', 1);
    $find_result = $find_request->execute();
    if (FileMaker::isError($find_result)) {
        die ("Database Error");
    }

    // Store the category in the session.
    $_SESSION['categories'] = $find_result->getRecords();

}

// Grab the category info from the the session.
$categories = $_SESSION['categories'];
```

The comments in the code above help explain how the technique works, but here's a summary: We start by setting up the session -- using a session if it already exists, or creating a new one if necessary. We then check to see if we already have the category data stored in the session. If not, we query the FileMaker database and store the results in the session. And finally, we create a variable so that we can easily refer to the data more easily in the rest of our code.

To give you an idea of how significant an impact this technique can have on performance, let's first refer to the stats that I gave earlier. By changing our code to use the streamlined layout, we were able to reduce the size of the data returned by FileMaker from 1.2Mb to 268kb, and the time involved in retrieving the data from a half a second to a quarter of a second. Using the caching technique, after the first page is displayed, there's no additional data being retrieved from FileMaker, and very little time (.01 seconds to be exact) involved in pulling the data from the cache. And remember, we're getting the benefits of the cached data on every page that uses the data!

Please keep in mind that we're only able to use this technique because the data that we're caching doesn't change very often -- and when it does, it's okay that some users don't see the changes immediately. Data that changes often, where the changes are important, should not be cached. For example, we probably wouldn't want to cache product pricing or availability, as it is likely that data changes often, and it is critical that users always see up-to-date data.

And one final note about this technique: In the code above, we've used PHP sessions to cache the data, so we're creating a separate cache for each user. We could also have cached the data at the application level using temporary files.

With the two techniques above, you've now got an easy way to optimize the Web applications that you're developing with the FileMaker API for PHP. In the future, I'll share some additional techniques as well.

Tim Dietrich, is the Founder and Lead Developer of Xgravity, a Richmond, Virginia-based information technology firm that designs and develops custom database and Web solutions using FileMaker Pro. Tim has been using FileMaker since 1992, and is a FileMaker 9 Certified Developer. Learn more about Tim and Xgravity at: <http://www.xgravity.net>