

# Smart Caching Techniques

by Tim Dietrich

Over the past few months, I have discussed several techniques that can be used to optimize FileMaker-based Web applications. One technique that I've written about extensively is caching. Caching allows us to request data from the database and store it on the Web server for future use, thus reducing the number of requests that need to be sent to FileMaker Server and improving the performance of the application as a result.

## Review: Caching In Session State

In September, I presented a method for caching data in session state. With that method, a cache is created for each user of the application, and is ideal when the data being cached is user-specific (a user's profile, for example).

Here's a code snippet showing how that 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'];
```

## Review: Caching With Temporary Files

In December, I presented another caching technique -- a method for storing cached data in temporary files on the server. With that method, the cache can be shared by all users of the application, and is ideal when the data being cached applies to all users (a list of categories or products in an online store, for example).

Here's code showing how that technique works:

```
<?php
```

```
// Include the FileMaker API for PHP.
require_once ("FileMaker.php");

// If a cache file exists and can be read...
if ( $categories_cache = @file_get_contents ("cache/categories.txt") ) {

    // Load the class definition "FileMaker_Result" to avoid receiving the error:
    // "The script tried to execute a method or access a property of an incomplete object."
    require_once ("Result.php");

    // Unserialize the cached find result.
    $find_result = unserialize ($categories_cache);

} else {

    // Get the database connection information.
    require_once ("database_settings.php");

    // Create a connection to the database.
    $fm = new FileMaker ();
    $fm->setProperty('hostspec', FM_HOSTSPEC);
    $fm->setProperty('database', FM_FILE);
    $fm->setProperty('username', FM_USERNAME);
    $fm->setProperty('password', FM_PASSWORD);

    // Find categories that are online, and sort them by name.
    $find_request = $fm -> newFindCommand ("PHP - Categories");
    $find_request -> addFindCriterion ("Is_Online", "Yes");
    $find_request -> addSortRule ('Category_Name', 1);
    $find_result = $find_request -> execute();

    // If FileMaker encountered an error...
    if (FileMaker::isError ($find_result)) {
        echo "Error Code: " . $find_result -> getCode() . "<br>";
        echo "Error Message: " . $find_result -> getMessage() . "<br>";
        die;
    } else {
        // Save a serialized version of the find results in the cache.
        file_put_contents ("cache/categories.txt", serialize ($find_result));
    }

}

// Get the records from the find result.
$categories = $find_result->getRecords();

// Display the categories.
foreach ( $categories as $category ) {
    echo $category -> getField("Category_Name") . "<br>";
}

}
```

```
?>
```

## Caching: Pros and Cons

With those two caching techniques, you can greatly reduce the number of requests that the Web application sends to FileMaker, and you can significantly boost the performance of your application as a result.

However, one issue with caching is that cached data can get stale. Continuing with the example above (where we cached categories for an online store), suppose that the store administrator takes one of those categories offline. If the categories were cached prior to the category being taken offline, then the cache will still include that category.

In this issue, I'll present a way to automatically refresh the cache when it ages.

### “Smart” Caching In Session State

As I mentioned above, the challenge with caching is that the data in the cache can become stale. To prevent this, we need to keep track of when the data was cached, and then, prior to using data in the cache, check to see how old it is. If the cache is still “fresh,” then we use it. However, if the cache has expired, then we need to dump and reload it. I refer to this as “smart” caching -- where the code is smart enough to know when to use -- and not use -- the cache.

Here is sample code that shows smart caching where the cache is stored in session state:

```
<?php

// Include the FileMaker API for PHP.
require_once ("FileMaker.php");

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

// Specify the age (in seconds) of the cache before it expires.
// In this case, the cache is set to expire in 5 minutes.
$cache_age_max = 60 * 5;

// Assume that we will not have cached data to work with.
$find_result = null;

// If we have cached data in the session state...
if (isset($_SESSION['categories'])) {

    // Get the age of the cached data.
    $cache_age = (time () - $_SESSION['categories_cache_time']);

    // If the cache hasn't expired...
    if ($cache_age < $cache_age_max) {

        // Load the class definition "FileMaker_Result" to avoid the error:
```

```

        // "The script tried to execute a method or access a property of an incomplete object."
        require_once ("Result.php");

        // Unserialize the cached find result.
        $find_result = unserialize ($_SESSION['categories']);
    }
}

// If we weren't able to get the data from the cache...
if ($find_result == null) {

    // Get the database connection information.
    include ("database_settings.php");

    // Create a connection to the database.
    $fm = new FileMaker ();
    $fm->setProperty('hostspec', FM_HOSTSPEC);
    $fm->setProperty('database', FM_FILE);
    $fm->setProperty('username', FM_USERNAME);
    $fm->setProperty('password', FM_PASSWORD);

    // Find only categories that are online, and sort them by name.
    $find_request = $fm -> newFindCommand ("PHP - Categories");
    $find_request -> addFindCriterion ("Is_Online", "Yes");
    $find_request -> addSortRule ('Category_Name', 1);
    $find_result = $find_request -> execute();

    // If FileMaker encountered an error...
    if (FileMaker::isError ($find_result)) {
        echo "Error Code: " . $find_result -> getCode() . "<br>";
        echo "Error Message: " . $find_result -> getMessage() . "<br>";
        die;
    } else {

        // Save the find results in the cache.
        $_SESSION['categories'] = serialize ($find_result);

        // Record the time that the results were cached.
        $_SESSION['categories_cache_time'] = time ();
    }
}

// Get the records from the result.
$categories = $find_result->getRecords();

// Display the categories.
foreach ( $categories as $category ) {
    echo $category -> getField("Category_Name") . "<br>";
}

```

?>

The code has been thoroughly commented. However, let's review some important aspects of it:

- First, I am setting a variable called `$cache_age_max` that is used to indicate how old cached data can be before it expires. In this case, I'm specifying that the cache expires in 5 minutes.
- Next, notice that I've modified the original code a bit so that we first check to see if the data exists in the cache. If it does, then we check to see if the cache is still fresh -- and we do this by taking the age of the cache (the current time minus the time that we cached the data) and comparing it to the value that we stored in `$cache_age_max`. If the data is fresh, then we unserialize the cached data, and we're ready to use it.
- And finally, I have also made changes to the code so that if we don't have cached data to use, then we get the data from the FileMaker database, we serialize the results in the session state, and we store the time that the data was cached in a new session variable called `$_SESSION['categories_cache_time']`.

## “Smart” Caching With Temporary Files

We now have a way to cache data in session state and ensure that it is “fresh.” Let's look at similar code that allows us to create a shared cache with temporary files, and to ensure that it is fresh as well:

```
<?php
```

```
// Include the FileMaker API for PHP.
require_once ("FileMaker.php");

// Specify the cache file to use.
$cache_file_path = "cache/categories.txt";

// Specify the age (in seconds) of the cache before it expires.
$cache_age_max = 60 * 5;

// Assume that we will not have cached data to work with.
$find_result = null;

// If the cache file exists...
if (file_exists ($cache_file_path) ) {

    // Get the age of the cache file.
    $cache_age = (time () - filemtime ($cache_file_path));

    // If the cache hasn't expired...
    if ($cache_age < $cache_age_max) {

        // Read the cache.
        $categories_cache = @file_get_contents ("cache/categories.txt");

        // Load the class definition "FileMaker_Result" to avoid the error:
        // "The script tried to execute a method or access a property of an incomplete object."
        require_once ("Result.php");
```

```

        // Unserialize the cached find result.
        $find_result = unserialize ($categories_cache);

    }

}

// If we weren't able to get the data from the cache...
if ($find_result == null) {

    // Get the database connection information.
    include ("database_settings.php");

    // Create a connection to the database.
    $fm = new FileMaker ();
    $fm->setProperty('hostspec', FM_HOSTSPEC);
    $fm->setProperty('database', FM_FILE);
    $fm->setProperty('username', FM_USERNAME);
    $fm->setProperty('password', FM_PASSWORD);

    // Find only categories that are online, and sort them by name.
    $find_request = $fm -> newFindCommand ("PHP - Categories");
    $find_request -> addFindCriterion ("Is_Online", "Yes");
    $find_request -> addSortRule ('Category_Name', 1);
    $find_result = $find_request -> execute();

    // If FileMaker encountered an error...
    if (FileMaker::isError ($find_result)) {
        echo "Error Code: " . $find_result -> getCode() . "<br>";
        echo "Error Message: " . $find_result -> getMessage() . "<br>";
        die;
    } else {
        // Save the find results in the cache.
        file_put_contents ("cache/categories.txt", serialize ($find_result));
    }

}

// Get the records from the result.
$categories = $find_result->getRecords();

// Display the categories.
foreach ( $categories as $category ) {
    echo $category -> getField("Category_Name") . "<br>";
}

?>

```

This code is very similar to the code that we used to store the cache in session state. The differences are subtle, yet important. Since we are using a file to store the cached data, we need an alternative way to get the age of the cache. PHP's `filemtime` and `time` functions allow us to do just that. By subtracting the value returned from the `time` function (the current time) from the value returned by the `filemtime` function (the time that the cache file was last modified), we get the number of seconds that have elapsed since the file was created. We use

that value to see if the cache is fresh, and if so, we use it. If the cache has expired, then we get fresh data from FileMaker, and save it in the cache.

## Summary

With the techniques that I've presented here, you now have several ways to safely cache data and reduce the number of requests that your Web applications send to FileMaker. This should result in applications that are much more efficient and faster.

In a future issue, I'll wrap up our discussion on optimization by presenting techniques for reducing the volume of the data that FileMaker returns.

Until then, happy coding!

**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>**