FileMaker for PHP Developers

FileMaker is a popular and powerful desktop database application toolkit. Recently, FileMaker, Inc. released a beta version of the FileMaker API for PHP, which allows PHP to more easily talk to the FileMaker Server Advanced product. Learn how to leverage FileMaker's strengths to deliver complex Web applications in a fraction of the time it would take using a typical SQL database.

by Jonathan Stark

PHP: *4.3.x or better*

O/S: *Any supported by PHP*

Other Software: *FileMaker Pro and FileMaker Server Advanced*

TO DISCUSS THIS ARTICLE VISIT:
http://forum.phparch.com/358

If you are not familiar with FileMaker (or have not checked it out in a while), this article should give you a good feel for the Web publishing capabilities of a unique database application that is installed on 12 million computers around the globe.

I have two goals for this article:

- To familiarize you with the FileMaker environment
- To introduce you to the FileMaker API for PHP

What is FileMaker?

FileMaker is a workgroup productivity toolkit designed to let knowledge workers quickly and easily construct data management systems for themselves. It is often referred to as a database—which it is—but it is more than that. In order to give you a solid understanding of how and when to use FileMaker, I am going to run through a real-world scenario.

Spreadsheet Headaches

ABC Company sells office supplies. Their catalog team is responsible for publishing product catalogs that are distributed to the sales force for generating new business. Every week or so, the catalog team receives spreadsheets of products that need to be included in upcoming catalogs. Information and images for any new products must be procured from the product manufacturer. Once all the data has been gathered, it is entered into a page layout program and printed.The catalog team finds this process very frustrating, because the spreadsheets are more often similar than not. Any given item might be going into multiple catalogs, so any changes to the item price, for example, must be made in multiple spreadsheets.

Even worse, the team members have to pass documents back and forth, making sure to keep track of who has the most recent version. It is all too easy to overwrite the wrong file, or to find different changes made to two copies of the same document. Merging these changes is a tedious manual process.

Clearly, the team needs a database application, but the IT department is too busy to deal with it. The catalog team know exactly what they need, and would be happy to build it themselves, but none of them have the time (or desire) to become a programmer.

FileMaker to the Rescue

One of the catalog designers—I'll call him Dave—does a little research and stumbles upon FileMaker Pro. He downloads a trial version and, in a couple of hours, has all his catalog spreadsheets imported into a single relational database file. FileMaker Pro has layout and scripting tools built right in, and Dave is able to rough out a series of data entry screens and buttons that perform automated tasks. It's nothing fancy, but it works.Within a week, Dave's boss Cathy notices that Dave is cranking out catalogs with time to spare. She asks him what his secret is, and he shows her his little database application. She likes what she sees of the database application and asks Dave what it would take to roll it out to the five other catalog designers. He tells her that she just needs to purchase and install the software. The other designers can actually log into Dave's database directly from their machines using FileMaker Pro's peer-to-peer sharing feature. They would all be working in the same database at the same time—no more emailing spreadsheets back and forth. It just so happens that it is the end of the fiscal year and Cathy has some leftover budget burning a hole in her pocket, so she buys five sets of FileMaker Pro.

This all works out great. All the designers are happy that the most tedious part of their job has evaporated, and Cathy is happy because the team's output and accuracy is at an all time high.

Now Everyone Wants FileMaker

Interestingly, Dave now finds himself spending more time improving the database than building catalogs. The team continually suggests tweaks and improvements that he is only too happy to implement. The application starts to get quite sophisticated and has lots of embedded business logic. More importantly, it is becoming the best source of up-to-date product information in the entire organization.Dave's little database starts to gain notoriety. Product managers and salespeople start calling Dave for custom reports. Cathy's boss takes notice of the team's increased output and is given a tour of the application. Before you know it, the marketing department wants a version. Dave is happy to comply and builds the features that they need. However, there are quite a few people in the marketing department, and FileMaker Pro can only share peer-to-peer with five users.

The IT department is called in and they purchase a copy of FileMaker Server, which allows up to 250 connections. The server software is installed and Dave's little database is now hosted to anyone in the company that has FileMaker Pro installed.

At this point, Dave is busier than ever. People are constantly calling on him for custom reports and new features. At the same time, there is a growing desire from the sales force to get access to the database from outside the office, preferably via the Web. If they could just browse the product data without having FileMaker Pro installed, they could stop pestering Dave.

Dave suggests to IT that they upgrade FileMaker Server to FileMaker Server Advanced (FMSA), because FMSA allows Web connections. IT gets approval and the server is upgraded. Unfortunately, Dave doesn't know PHP.

That's where you come in.

Pause for Perspective

This example is not meant to sell you on FileMaker. It is meant to illustrate how quickly and thoroughly it can penetrate a small business or a workgroup. If Dave had had to build the application with PHP and MySQL, it would never have been built. Even if he knew how to do it that way—which he didn't—it would have taken him ten times as long to build.This increased development time is significant. When application development takes a long time, it is common for the business needs to have changed in the meantime. The marketplace is not going to stand still while you are coding. Rapid change is the hallmark of FileMaker applications. When the business changes fast, FileMaker is a great option for internal systems.

When Web development needs to keep pace with these rapid change requests, it can present a problem. The only way you can realistically pull it off is to have the Web site inherit much of the business logic that is built into the internal system—which brings us neatly to the FileMaker API for PHP.

The FileMaker API for PHP

You can connect to FileMaker Server Advanced via ODBC, but to do so with a Web application would be to discard the main advantage of using FileMaker as your back-end—namely, the reuse of embedded business logic. Fortunately, FileMaker, Inc. has released a beta of something called the FileMaker API for PHP. It is a bundle of object oriented PHP files that you can install on any typical Web server running PHP 4.3.x or greater. The API is a free download, although using it requires FileMaker Server Advanced and FileMaker Pro, which are not free.The bundle has one main file called—appropriately enough—**FileMaker.php**. If you want to talk to FileMaker with PHP, you just download the bundle, include **FileMaker.php** in the page you are working on, point it

at FMSA, and you are off to the races.

As an aside, there is also an installer version of the API available—but if you are already running PHP, you are going to want to avoid it because it will override your **php.ini** file.

The developers of the API went to great lengths to allow access to more than just the data in the database—they knew that much of the 'data' in a FileMaker solution is embedded in the interface itself, so they revealed it to the API. As a result, connecting to FileMaker using the API for PHP allows you to run scripts, pull value lists from your layouts, work with portals of related records, and so on.

Of course, none of this carries much weight if you don't know what portals, scripts, and value lists are, or how incredibly easy it is to make them. My next step is to give you some idea.

No Habla FileMaker?

It is beyond the scope of this article to teach you how to develop FileMaker databases, but understanding the terminology will be a great start, so let's begin with a glossary of FileMaker terms:

- *File*—A FileMaker file is a single self-contained binary that is created by the FileMaker Pro application. FileMaker files are sometimes referred to simply as 'databases', and can be identified by their **.fp7** extension. Like a typical database, they can contain one or more tables. Unlike a typical database, they can also contain a UI for interacting with the data in the tables.
- *Window*—When you open a FileMaker file with FileMaker Pro, a window is created. You can have multiple windows into a given file, but you must have at least one window open for the file to be open. When you close the last window into a file, the file closes.
- *Layout*—A layout is what you is displayed in a FileMaker window. A layout is basically a screen that allows users to search, view, and edit data in the file. Each layout is linked to a single underlying table. Each table can have any number of layouts attached to it. Layouts come in three flavors: **Table view**, which shows your records in a spreadsheet style format, **List view**, which shows multiple records in the same window (think of Google search results), and **Form view**, which shows a single record at a time (think of a typical Web form).
- *Mode*—When you are viewing a layout in a window, you can opt to view it in one of four modes: **Find**, **Browse**, **Preview** or **Layout**. A user would enter **Find** mode to define search criteria for records. To create, edit, duplicate, or delete records, you must be in **Browse** mode. Entering **Preview** mode will show you how the layout will look if printed. **Layout** mode is for creating, editing, duplicating or deleting the objects on a layout, or the layouts themselves.
- *Field*—The word 'field' can be quite confusing in FileMaker because it means two different things depending on the context. If

In LESS Than ONE MINUTE You Can Run the Easiest, Best-Guaranteed, Re-brandable PHP Email Management Software On the Planet and Triple Your Email Marketing Success Rates!

You may have tried other email systems or auto-responder software, or even tried programming your own email software yourself. STOP!

This time, try the Rolls-Royce of PHP Email - oemPro:

- Manage newsletters with Ease
- User friendly
- Track click-throughs better, faster
- Send in any format

Discover how people just like you are producing high quality online newsletters, managing lists and maintaining customer information without all the headaches and hassles. Try it RISK FREE today, go to www.octeth.com


oemPro The International Rolls-Royce of PHP Email marketing software

a product of octeth

LISTING 1

```php
<?php

# For security reasons, these lines should either be included from a
# config file above the Web directory, or possibly captured during a
# login and stored in the SESSION superglobal array
define('FM_HOST', '127.0.0.1');
define('FM_FILE', 'ProductCatalog.fp7');
define('FM_USER', 'esmith');
define('FM_PASS', 'f!r3crack3r');

# grab search criteria, if any has been sent
$criteria = (array_key_exists('criteria', $_GET)) ? htmlspecialchars($_GET['criteria']) : '';

# grab the sort column, if any has been sent
$column = (array_key_exists('column', $_GET)) ? htmlspecialchars($_GET['column']) : '';

# set the layout name for this page
$layout_name = 'view_products';

# set convenience var
$this_page = $_SERVER['PHP_SELF'];

# initialize our output var
$page_content = '';

# this is the include for the API for PHP
require_once ('FileMaker.php');
# instantiate a new FileMaker object
$fm = new FileMaker(FM_FILE, FM_HOST, FM_USER, FM_PASS);

# get the layout as an object
$layout_object = $fm->getLayout($layout_name);

# check for errors
if (FileMaker::isError($layout_object)) {
    die('<p>'.$record->getMessage().' (error '.$record->code.')</p>');
}

# get the fields as an array of objects
$field_objects = $layout_object->getFields();

# create a new search transaction
$request = $fm->newFindCommand($layout_name);

# indicate that we want an OR search
$request->setLogicalOperator(FILEMAKER_FIND_OR);

# search each field on the layout for the criteria, if any
#
# NOTE: I am using the getResult method of the field object to check
# the data type of the field. Even in a find request, data type
# formatting must be respected. If we didn't check for this, we
# would get an error if we searched a date field for the value
# 'Erica', for example
foreach($field_objects as $field_object) {
    $field_name = $field_object->getName();
    # format the criteria appropriately for the current field data type
    if ($field_object->getResult() == 'date') {
        if (strtotime($criteria)) {
            $request->addFindCriterion($field_name, date('n/j/Y', strtotime($criteria)));
        }
    } elseif ($field_object->getResult() == 'time') {
        if (strtotime($criteria)) {
            $request->addFindCriterion($field_name, date('H:i:s', strtotime($criteria)));
        }
    } elseif ($field_object->getResult() == 'timestamp') {
        if (strtotime($criteria)) {
            $request->addFindCriterion($field_name, date('n/j/Y H:i:s', strtotime($criteria)));
        }
    } elseif ($field_object->getResult() == 'container') {
        # skip this field because it is a container (like a blob) and can't be searched for text
    } else {
        $request->addFindCriterion($field_name, $criteria);
    }
}

# specify sort column (aka, field), if any
$request->addSortRule($column, 1);

# execute the search transaction
$result = $request->execute();
```

LISTING 1: *Continued...*

```php

# check for errors (including no records found)
if (FileMaker::isError($result)) {
    die('<p>'.$record->getMessage().' (error '.$record->code.')</p>');
}

# display the found count
$total = $result->getTableRecordCount();
$found = $result->getFoundSetCount();
$s = ($found==1) ? '' : 's';
$page_content .= '<p>Your search for "'.$criteria.'" returned '.$found." record{$s} of ".$total.' total</p>';

# get the result record set as an array of record objects
$record_objects = $result->getRecords();

# start compiling our record output
$page_content .= '<table border="1">';
$page_content .= '<tr>';
$page_content .= '<th> </th>';

# loop through array of field objects to draw header
foreach($field_objects as $field_object) {
    $field_name = $field_object->getName();
    $page_content .= '<th><a href="'.$this_page.'?criteria='.$criteria.'&column='.$field_name.'">'.$field_name.'</a></th>';
}
$page_content .= '</tr>';

# loop through record objects
foreach ($record_objects as $record_object) {
    $page_content .= '<tr>';
    $page_content .= '<td><a href="view_product.php?recid='.$record_object->getRecordId().'">view</a></td>';

    # loop through array of field objects
    foreach($field_objects as $field_object) {
        $field_name = $field_object->getName();
        $field_val = $record_object->getField($field_name);
        $field_val = htmlspecialchars($field_val, ENT_QUOTES);
        $field_val = nl2br($field_val);
        $page_content .= '<td>'.$field_val.'</td>';
    }
    $page_content .= '</tr>';
}
$page_content .= '</table>'."\n";

?>
<html>
<head>
<meta http-equiv="Content-type" content="text/html; charset=utf-8">
<title>view_products</title>
<style type="text/css" media="screen">
body {font: 75% "Lucida Grande", "Trebuchet MS", Verdana, sans-serif;}
table {width: 600px;border-collapse:collapse;border-color: #cccccc;}
th {padding: 3px; background-color: #DDD; text-align: center;}
td {padding: 3px;}
a, a:visited {color: blue;text-decoration: none;font-weight: bold;display: block;}
a:hover, a:active {color: blue;text-decoration: underline;font-weight: bold;}
</style>
</head>
<body id="view_products" onload="">
<form action="<?php echo $this_page ?>" method="get">
<p>
<input type="text" name="criteria" value="<?php echo $criteria; ?>" />
<input type="submit" value="search" />
</p>
</form>
<?php echo $page_content; ?>
</body>
</html>

```

you are defining a table, the word 'field' is equivalent to the word 'column' in traditional SQL databases. So, you might ask a FileMaker developer to "Add a Phone Number field to the Company table." However, the word 'field' means something very different when you are defining a layout in layout mode. On a layout, a field is a like a form input on a Web page. It is a rectangular area that allows a user to interact with a particular cell of data—a 'field' of a particular record. When I am teaching my Intro to FileMaker class, I often explicitly refer to **Table Fields** and **Layout Fields** until students start to get the feel for things.

- *Value List*—A value list is simply a return delimited list of values, usually used to aid data entry of common values such as US state abbreviations. A value list can be applied to a field on a layout to aid data entry. When you attach a value list to a layout field, you specify the way you want it to look. It can be formatted to display as a pop-up menu, a drop-down list, radio buttons, or checkboxes. Each behaves more or less like similarly named HTML form controls.
- *Portal*—As mentioned above, layouts are tied to a particular table. Often, you want to display data that is related to a given record—for example, showing a product record and viewing the related inventory data. This is done with a layout object called a portal. You can think of it as a list view embedded in a form view.
- *Script*—FileMaker has a built-in macro scripting language. It has a point and click interface that is accessed by selecting ScriptMaker under the Scripts menu. Once a script is created, it is a trivial matter to attach it to an object on a layout, thereby turning the object into a button. When the button is clicked in **Browse** or **Find** mode, the script runs.

A Picture is Worth a Thousand Words

Now that you know what everything is called, let's look at a couple of FileMaker layouts. By the way, these screenshots are of the **ProductCatalog.fp7** file that is included with the FileMaker API for PHP download bundle.Figure 1 is a list view layout. At the top of the layout is the header, which contains a number of buttons that trigger scripts when clicked. Beneath the header is the body, which contains a list of records. If the number of records is such that they will not all fit in the window, the body section can be scrolled while the header remains in view at the top of the window.

Figure 2 is a form view layout. Like the list view, it has a header area with buttons, but in the body part of the layout, we are looking at a single record. Notice that the category field has a value list applied to it as checkboxes. At the very bottom of the layout, labeled with the word **Inventory**, is a portal. As described above, it is displaying a list view of records from the **Inventory** table that are related to the current product record.

Bear in mind while reviewing these layouts that even a novice FileMaker user could edit, add, delete, or rearrange these objects in a matter of minutes. And—for better or worse—they often do. In fact, much of the time spent working on a FileMaker solution is spent on the layouts.

When you are publishing to the Web with FileMaker, you don't talk to the tables, directly.

Why You Should Care

This is probably going to sound counter-intuitive at first, but when you are publishing to the Web with FileMaker, you don't talk to the tables directly—you talk to layouts. Other than a couple of minor exceptions, all calls to a FileMaker database from PHP include a layout name. When you query the database, you only get back the fields that are present on the layout that you targeted—you don't get all the fields from the underlying table.At first, this drove me nuts. I just wanted to access the data in the table directly and on the fly. For example, if I wanted to express something like:

```
SELECT Name, Description FROM Product WHERE ID=1;
```

I would have to launch FileMaker Pro, open the file, make a layout with **Name**, **Description**, and **ID** fields on it, and point my code at that layout. What this meant was that I was constantly going back and forth between the FileMaker Pro application and my PHP text editor in order to write a page.

However, when I started working on real-world solu-

tions for customers, I came to appreciate this arrangement. As I took pains to illustrate above, FileMaker customers usually have lots of business logic embedded in their layouts, and they are constantly modifying things. The first time I *didn't* have to recreate all that layout logic in PHP, I saw the light. That is because FileMaker allows you to pull information about the layout itself in the query. A lot of time goes into creating layout objects like portals, value lists and scripts. Those things can be very usefully reused on a Web page, so why reinvent the wheel?

Alright, Already, Let's See an Example

What I am going to do here is write a couple of pages that take most of their cues from the two FileMaker layouts that we have seen so far. The one in Listing 1 will be called **view_products.php** and will correspond to a table view layout in FileMaker, and the script in Listing 2 will be **view_product.php**, which will correspond to a form view layout.The pages were written specifically to handle updates to the layouts they are attached to. In fact, they can be pointed at any FileMaker layout, and they will do a pretty good job of rendering the information on the Web as it is displayed in FileMaker. For example, I could put these pages on a Web server; my customer could then add or delete fields from a layout, and their changes would auto-magically appear on the site.

Explanations are included as comments inline with the code. There is nothing really earth shattering here—it

FIGURE 1

FIGURE 2

FIGURE 3

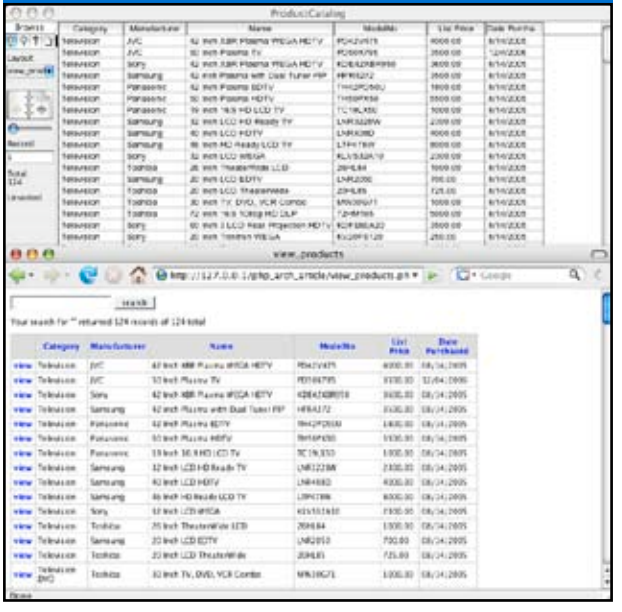
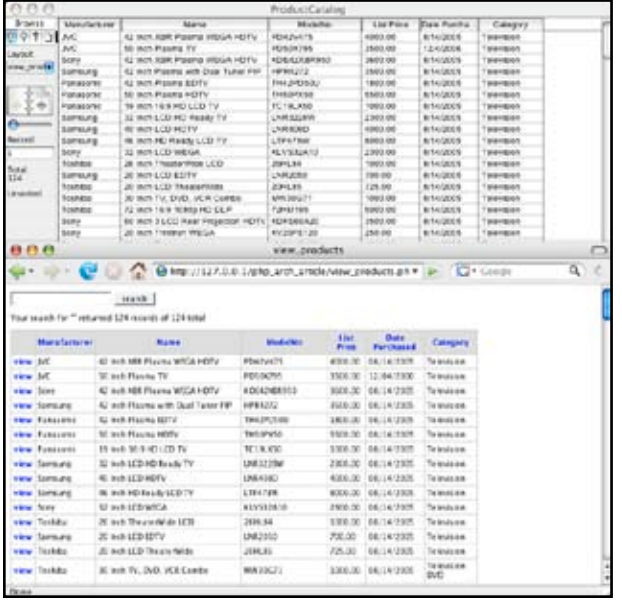


FIGURE 4



is really just an introduction to some of the more useful objects and methods in **FileMaker.php**.

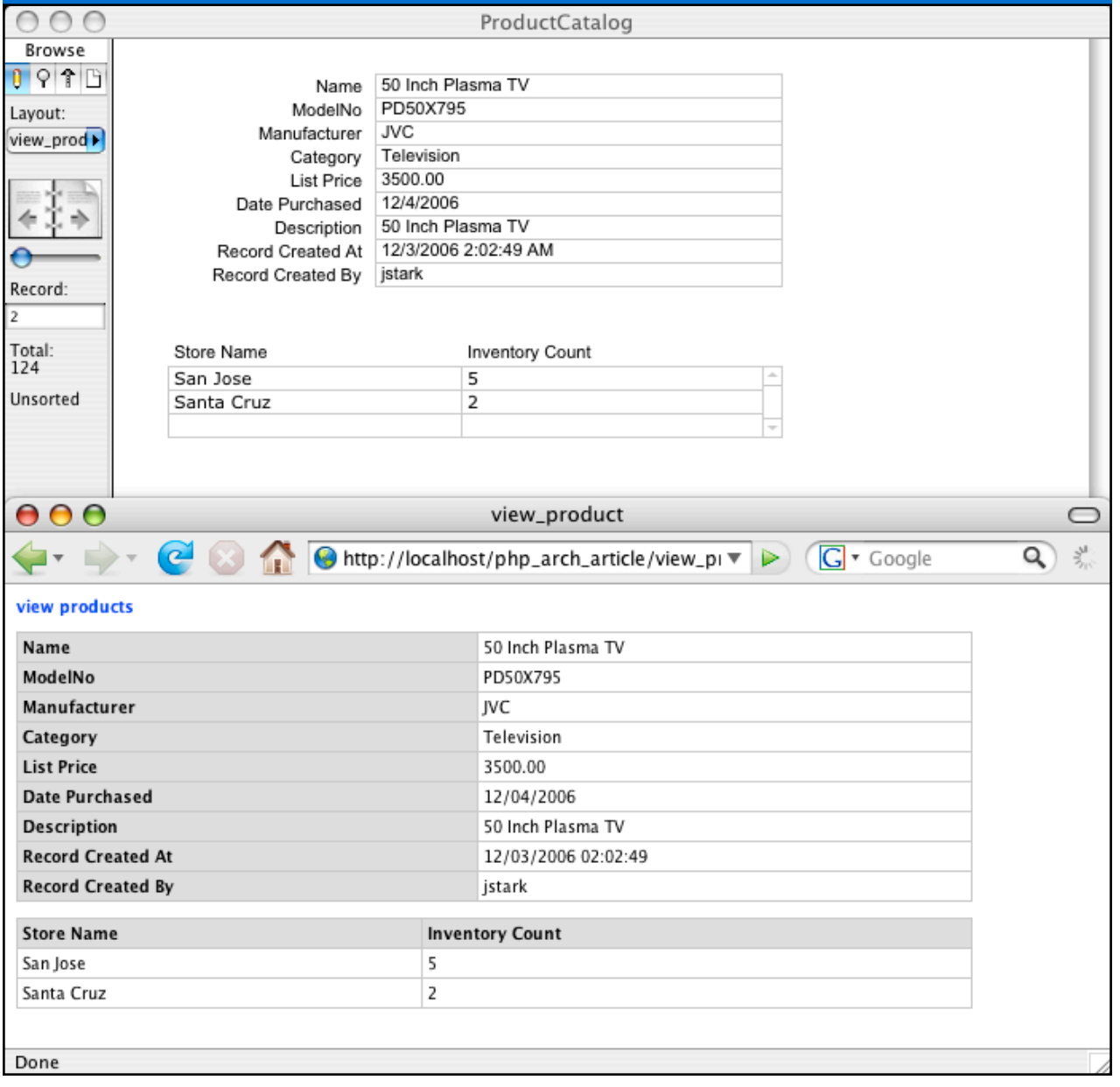
View_products.php

Refer to Figure 3, and you will notice the similarity between the FileMaker layout and the resulting Web page. The underlying FileMaker layout is at the top of the picture, and the browser version is directly beneath it. Notice, when you are reading through the following code, that I never refer to any fields by name—they show up on the Web (and are searchable) solely by virtue of the fact that they are on the layout. Compare Figure 3 with Figure 4 to see what happens to the Web page when I reorder the fields on the FileMaker layout.Form View

With the form view, there is again a strong similarity between the FileMaker layout and the resulting Web page; refer to Figure 5. The underlying FileMaker layout is at the top of the picture, and the browser is directly beneath it. As with the previous example, I never refer to any fields by name—they just show up on the Web (and are searchable) solely by virtue of the fact that they are on the layout. This time, we add a portal that contains related records from the Inventory table.RAD Comes at a Price

Using FileMaker as a Web backend can allow you to rapidly develop and deploy powerful and complex solu-

FIGURE 5



tions because it allows you to reuse existing business logic. However, this advantage does have a price. Since FileMaker is sending back so much information in addition to the actual record data, it is not as fast as a typical SQL database. Also, there is a hard limit of 100 maximum concurrent connections to the database. Fortunately, most connections last less than a second, so in practice it is not uncommon to serve data to thousands of browsers at the same time because it is unlikely that they would all refresh their user agents simultaneously. All things considered, FileMaker is well suited for intranet or extranet style solutions, where the user group is a known quantity and a login is required.Conclusion

Well, here I am at the end of the article, and we have so much more to discuss. Hopefully, you now have a basic feel for how to best use FileMaker as a Web backend. Stay tuned for Part 2 of this series, when you'll learn how to use value lists and scripts pulled from FileMaker to edit records.

LISTING 2

```
<?php

# For security reasons, these lines should either be included from a
# config file above the Web directory, or possibly captured during a
# login and stored in the SESSION superglobal array
define('FM_HOST', '127.0.0.1');
define('FM_FILE', 'ProductCatalog.fp7');
define('FM_USER', 'esmith');
define('FM_PASS', 'f!r3crack3r');

# grab the record id sent in the url
$recid = (array_key_exists('recid', $_GET)) ? htmlspecialchars($_GET['recid']) : '';

# set the layout name for this page
$layout_name = 'view_product';

# initialize our output var
$page_content = '';

# this is the include for the API for PHP
require_once ('Filemaker.php');

# instantiate a new FileMaker object
$fm = new FileMaker(FM_FILE, FM_HOST, FM_USER, FM_PASS);
# get the record by its id
$record = $fm->getRecordById($layout_name, $recid);

# check for errors
if (FileMaker::isError($record)) {
    die('<p>'.$record->getMessage().' (error '.$record->code.')</p>');
}

# get the layout as an object
$layout_object = $record->getLayout();

# get the fields from the layout as an array of objects
$field_objects = $layout_object->getFields();

# start compiling our output
$page_content .= '<table border="1">';
foreach($field_objects as $field_object) {
    $field_name = $field_object->getName();
    $field_value = $record->getField($field_name);
    $field_value = htmlspecialchars($field_value, ENT_QUOTES);
    $field_value = nl2br($field_value);
    $page_content .= '<tr><th>'.$field_name.'</th><td>'.$field_value.'</td></tr>';
}
$page_content .= '</table>'."\n";

# check the layout for portals
$portal_objects = $layout_object->getRelatedSets();

foreach($portal_objects as $portal_object) {
    $page_content .= '<table border="1">';
    # loop through the portal fields to draw the table header row
    $page_content .= '<tr>';
    $field_names = $portal_object->listFields();
    foreach($field_names as $field_name) {
        # related fields are returned with double colons in the name, so remove them
        $page_content .= '<th>'.str_replace('::', ' ', $field_name).'</th>';
    }
    $page_content .= '</tr>';

    # get the name of the current portal object
    $portal_name = $portal_object->getName();

    # get the records related to this record, based on the portal name
    $related_records = $record->getRelatedSet($portal_name);

    # if there are no related records in the portal, filemaker will return an error
    if (FileMaker::isError($related_records)) {
        $page_content .= '<td colspan="'.count($field_names).'">no related records</td>';
    } else {
        foreach($related_records as $related_record) {
            foreach($field_names as $field_name) {

                $field_val = $related_record->getField($field_name);
                $field_val = htmlspecialchars($field_val, ENT_QUOTES);
                $field_val = nl2br($field_val);
                $page_content .= '<td>'.$field_val.'</td>';
            }
            $page_content .= '</tr>';
        }
    }
    $page_content .= '</table>'."\n";
}
?>
<html>
<head>
<meta http-equiv="Content-type" content="text/html; charset=utf-8">
<title>view_product</title>
<style type="text/css" media="screen">
body {font: 75% "Lucida Grande", "Trebuchet MS", Verdana, sans-serif;}
table {width: 600px; border-collapse:collapse; border-color: #cccccc; margin-bottom: 10px;}
th {padding: 3px; background-color: #DDD; text-align: left;}
td {padding: 3px;}
a, a:visited {color: blue;text-decoration: none;font-weight: bold;}
a:hover, a:active {color: blue;text-decoration: underline;font-weight: bold;}
</style>
</head>

<body id="view_product" onload="">
<p><a href="view_products.php">view products</a></p>
<?php echo $page_content; ?>
</body>
</html>
```

JONATHAN STARK is the President of Jonathan Stark Consulting, an IT consulting firm located in Providence, RI. He consults a variety of clients from the creative industry including Staples, Turner Broadcasting, and Ambrosi. He has spoken at the FileMaker Developers Conference, is a Certified FileMaker Developer, and teaches training courses in both FileMaker and Web publishing. Jonathan is reluctant to admit that he began his programming career more than 20 years ago on a Tandy TRS-80. For more information, please visit http://jonathanstark.com.