FileMaker® Server 12

Custom Web Publishing with PHP

© 2007–2012 FileMaker, Inc. All Rights Reserved.

FileMaker, Inc.
5201 Patrick Henry Drive
Santa Clara, California 95054

FileMaker and Bento are trademarks of FileMaker, Inc. registered in the U.S. and other countries. The file folder logo and the Bento logo are trademarks of FileMaker, Inc. All other trademarks are the property of their respective owners.

FileMaker documentation is copyrighted. You are not authorized to make additional copies or distribute this documentation without written permission from FileMaker. You may use this documentation solely with a valid licensed copy of FileMaker software.

All persons, companies, email addresses, and URLs listed in the examples are purely fictitious and any resemblance to existing persons, companies, email addresses, or URLs is purely coincidental. Credits are listed in the Acknowledgements documents provided with this software. Mention of third-party products and URLs is for informational purposes only and constitutes neither an endorsement nor a recommendation. FileMaker, Inc. assumes no responsibility with regard to the performance of these products.

For more information, visit our website at http://www.filemaker.com.

Edition: 01

Contents

Preface

About this guide

This guide assumes you are experienced with PHP, developing websites, and using FileMaker® Pro to create databases. You should understand the basics of FileMaker Pro database design, and should understand the concepts of fields, relationships, layouts, portals, and containers. For information about FileMaker Pro, see FileMaker Pro Help.

This guide provides the following information about Custom Web Publishing with PHP on FileMaker Server:

- what is required to develop a Custom Web Publishing solution using PHP
- how to publish your databases using PHP
- what web users need in order to access a Custom Web Publishing solution
- how to use the FileMaker API for PHP to obtain data from databases hosted by FileMaker Server

Important You can download PDFs of FileMaker documentation from http://www.filemaker.com/documentation. Any updates to this document are also available from the website.

The documentation for FileMaker Server includes the following information:

For information about	See
Installing and configuring FileMaker Server	*FileMaker Server Getting Started Guide* FileMaker Server Help
Instant Web Publishing	*FileMaker Instant Web Publishing Guide*
Custom Web Publishing with PHP	*FileMaker Server Custom Web Publishing with PHP* (this book)
Custom Web Publishing with XML	*FileMaker Server Custom Web Publishing with XML*
Installing and configuring ODBC and JDBC drivers, and using ODBJ and JDBC	*FileMaker ODBC and JDBC Guide*

Chapter 1
Introducing Custom Web Publishing

With FileMaker Server, you can publish your FileMaker database on the Internet or an intranet in these ways.

Instant Web Publishing: With Instant Web Publishing, you can quickly and easily publish your database on the web. You don't need to modify your database files or install additional software—with compatible web browser software and access to the internet or an intranet, web users can connect to your database to view, edit, sort, or search records, if you give them access privileges.

With Instant Web Publishing, the host computer must be running FileMaker Pro, FileMaker Pro Advanced, or FileMaker Server Advanced. The user interface resembles the desktop FileMaker Pro application. The web pages and forms that the web user interacts with are dependent on the layouts and views defined in the FileMaker Pro database. For more information, see *FileMaker Instant Web Publishing Guide*.

Static publishing: If your data rarely changes, or if you don't want users to have a live connection to your database, you can use static publishing. With static publishing, you export data from a FileMaker Pro database to create a web page that you can further customize with HTML. The web page doesn't change when information in your database changes, and users don't connect to your database. (With Instant Web Publishing, data is updated in a web browser window each time the browser sends a request to FileMaker Server.) For more information, see *FileMaker Instant Web Publishing Guide*.

Custom Web Publishing: For more control over the appearance and functionality of your published database, use the Custom Web Publishing technologies available with FileMaker Server. FileMaker Server, which hosts the published databases, does not require FileMaker Pro to be installed or running for Custom Web Publishing to be available.

With Custom Web Publishing, you can:

- Integrate your database with another website
- Determine how users interact with data
- Control how data displays in web browsers

FileMaker Server provides two Custom Web Publishing technologies:

- Custom Web Publishing with PHP: Use the FileMaker API for PHP, which provides an object-oriented PHP interface to FileMaker Pro databases, to integrate your FileMaker data into a PHP web application. Because you code the PHP web pages yourself, you have complete control over the user interface and how the user interacts with the data.
- Custom Web Publishing with XML: Use XML data publishing to exchange FileMaker data with other websites and applications. By using HTTP URL requests with FileMaker query commands and parameters, you can query a database hosted by FileMaker Server, download the resulting data in XML format, and use the resulting XML data in whatever way you want.

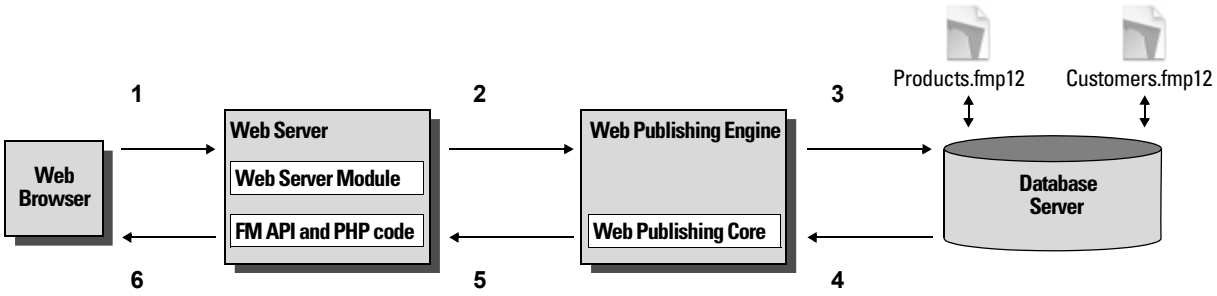
About the Web Publishing Engine

To support Instant Web Publishing and Custom Web Publishing, FileMaker Server uses a set of software components called the *FileMaker Server Web Publishing Engine*. The Web Publishing Engine handles interactions between a web user's browser, your web server, and FileMaker Server.

Custom Web Publishing with XML: Web users access your Custom Web Publishing solution by clicking an HREF link or by entering a Uniform Resource Locator (URL) that specifies the web server address and a FileMaker query string request. The Web Publishing Engine returns the XML data specified in the query string request.

Custom Web Publishing with PHP: When a web user accesses your Custom Web Publishing solution, PHP on FileMaker Server connects with the Web Publishing Engine and responds through the FileMaker API for PHP.

Using the FileMaker Server Web Publishing Engine for Custom Web Publishing



How a Web Publishing Engine request is processed

1. A request is sent from a web browser or application to the web server.
2. The web server routes the request through FileMaker's Web Server Module to the Web Publishing Engine.
3. The Web Publishing Engine requests data from the database hosted by the Database Server.
4. The FileMaker Server sends the requested FileMaker data to the Web Publishing Engine.
5. The Web Publishing Engine converts the FileMaker data to respond to the request.
   - For PHP requests, the Web Publishing Engine responds to the API request.
   - For XML requests, the Web Publishing Engine sends XML data directly to the web server.
6. The web server sends the output to the requesting web browser or program.

Important Security is important when you publish data on the web. Review the security guidelines in *FileMaker Pro User's Guide*, available as a PDF file from http://www.filemaker.com/documentation.

Custom Web Publishing with PHP

The FileMaker API for PHP provides an object-oriented PHP interface to FileMaker databases. The FileMaker API for PHP enables both data and logic stored in a FileMaker Pro database to be accessed and published on the web, or exported to other applications. The API also supports complex and compound find commands for extracting and filtering data stored in FileMaker Pro databases.

Originally designed as a procedural programming language, PHP has been enhanced as an object-oriented web development language. PHP provides programming language functionality for constructing virtually any type of logic within a site page. For example, you can use conditional logic constructs to control page generation, data routing, or workflow. PHP also provides for site administration and security.

Custom Web Publishing with XML

FileMaker Custom Web Publishing with XML enables you to send query requests to a FileMaker Pro database hosted by FileMaker Server, and display, modify, or manipulate the resulting data. Using an HTTP request with the appropriate query commands and parameters, you can retrieve FileMaker data as an XML document. You can then export the XML data to other applications.

Comparing PHP to XML

The following sections provide some guidelines for determining the best solution for your site.

Reasons to choose PHP

- PHP is a more powerful, object-oriented procedural scripting language, but is relatively easy to learn. There are many resources available for training, development, and support.
- The FileMaker API for PHP enables data and logic stored in a FileMaker Pro database to be accessed and published on the web, or exported to other applications.
- PHP lets you use conditional logic to control page construction or flow.
- PHP provides programming language functionality for constructing many types of logic on a site page.
- PHP is one of the most popular web scripting languages.
- PHP is an open source language, available at http://php.net.
- PHP enables access to a wide variety of third-party components that you can integrate into your solutions.

Reasons to choose XML

- FileMaker XML request parameter syntax is designed for database interaction, simplifying solution development.
- XML is a W3C standard.
- XML is a machine and human readable format that supports Unicode, enabling data to be communicated in any written language.
- XML is well-suited for presenting records, lists and tree-structured data.
- You can use FMPXMLRESULT for accessing XML data using Custom Web Publishing and for XML export from FileMaker Pro databases.

Note For more information about Custom Web Publishing with XML, see *FileMaker Server Custom Web Publishing with XML*.

Chapter 2
About Custom Web Publishing with PHP

Custom Web Publishing with PHP lets you use the PHP scripting language to integrate data from FileMaker databases with your customized web page layouts. Custom Web Publishing with PHP provides the FileMaker API for PHP, which is a PHP class created by FileMaker that accesses databases hosted by FileMaker Server. This PHP class connects to the FileMaker Server Web Publishing Engine and makes data available to your web server's PHP engine.

Key features in Custom Web Publishing with PHP

- Create web applications that use the Open Source PHP scripting language. Use the FileMaker Server supported version of PHP 5, or use your own version of PHP 5. (If you select to use your own version of PHP, see "Manually installing the FileMaker API for PHP" on page 11.)
- Host databases on FileMaker Server. FileMaker Pro is not required for Custom Web Publishing because FileMaker Server hosts the databases.
- Write PHP code that can create, delete, edit, and duplicate records in a hosted FileMaker database. Your code can perform field and record validation before committing changes back to the hosted database.
- Write PHP code that accesses layouts, portals, value lists, and related fields. Like FileMaker Pro, access to data, layouts, and fields is based on the user account settings defined in the database's access privileges. The Web Publishing Engine also supports several other security enhancements. See "Protecting your published databases" on page 14.
- Write PHP code that executes complex, multi-step scripts. FileMaker supports over 65 script steps in Custom Web Publishing. See "FileMaker scripts and Custom Web Publishing" on page 18.
- Write PHP code that performs complex find requests.

Custom Web Publishing requirements

This section explains what is required to develop a Custom Web Publishing solution using PHP, what web users need in order to access a Custom Web Publishing solution, and what impact hosting a web publishing solution may have on your server.

What is required to publish a database using Custom Web Publishing

To publish databases using Custom Web Publishing with PHP, you need:

- a FileMaker Server deployment, which includes three components.
  - a web server, either Microsoft IIS (Windows) or Apache (Mac OS). The FileMaker Web Server Module is installed on the web server.
  - the FileMaker Web Publishing Engine
  - the FileMaker Database Server

- PHP installed on the web server. FileMaker Server can install the supported version of PHP 5, or you can use your own version. The minimum required version of PHP on Mac OS X v10.6 is PHP 5.3.3. The minimum required version of PHP on Windows is PHP 5.3.5. For information about PHP, see http://php.net. The version of PHP installed on the web server must support cURL (client URL library) functions. For information about cURL, see http://php.net/curl.

  Important When you install the FileMaker Server supported version of PHP 5, it does not show up in the Mac OS X Server Admin tool; it is not supposed to be listed. If you use the Mac OS X Server Admin tool to turn on PHP, you disable the FileMaker Server supported version of PHP 5, and enable your own version of PHP.

- one or more FileMaker Pro databases hosted by FileMaker Server.
- the IP address or domain name of the host running the web server
- a web browser and access to the web server to develop and test your Custom Web Publishing solution

For more information, see *FileMaker Server Getting Started Guide*.

What web users need to access a Custom Web Publishing solution

To access a Custom Web Publishing solution that uses PHP, web users need:

- a web browser
- access to the Internet or an intranet and the web server
- the IP address or domain name of the host running the web server

If the database is password-protected, web users must also enter a user name and password for a database account.

Connecting to the Internet or an intranet

When you publish databases on the Internet or an intranet, the host computer must be running FileMaker Server, and the databases you want to share must be hosted and available. In addition:

- Publish your database on a computer with a full-time Internet or intranet connection. You can publish databases without a full-time connection, but they are only available to web users when your computer is connected to the Internet or an intranet.
- The host computer for the web server that is part of the FileMaker Server deployment must have a dedicated static (permanent) IP address or a domain name. If you connect to the Internet with an Internet service provider (ISP), your IP address might be dynamically allocated (it is different each time you connect). A dynamic IP address makes it more difficult for web users to locate your databases. If you are not sure of the type of access available to you, consult your ISP or network administrator.

Manually installing the FileMaker API for PHP

When you install FileMaker Server, you are given the option to install the FileMaker supported version of PHP (PHP 5). If you already have a PHP engine installed and configured and you want to add only the FileMaker API for PHP, then manually install the FileMaker API for PHP class to make it available to your PHP scripts.

If you did not install the FileMaker supported version of PHP, be sure to do the following configuration tasks on your version of the PHP engine:

- Enable the cURL module in php.ini.
- Specify the location of the FileMaker API for PHP in the include_path variable in php.ini.
- If you are accessing databases that contain dates and times, install the pear date package. For more information, see: http://pear.php.net/package/date/

Note FileMaker Server has been tested with with PHP version 5.3.3 for Mac OS X v10.6, and with PHP version 5.3.5 for Windows. For best results, use the appropriate version of PHP.

To make the FileMaker API for PHP accessible to your PHP scripts

When you installed FileMaker Server, the FileMaker API for PHP package was included as a .zip file in the following location:

- For IIS (Windows):
  `<drive>:\Program Files\FileMaker\FileMaker Server\Web Publishing\FM_API_for_PHP_Standalone.zip`
  where `<drive>` is the drive on which the web server component of your FileMaker server deployment resides.
- For Apache (Mac OS):
  `/Library/FileMaker Server/Web Publishing/FM_API_for_PHP_Standalone.zip`

The FM_API_for_PHP_Standalone.zip file contains a file called FileMaker.php and a folder called FileMaker. Unzip the file and copy the FileMaker.php file and the FileMaker folder to either of these locations:

- the web server root folder where your PHP scripts reside.
  - For IIS (Windows): `<drive>:\Inetpub\wwwroot` where `<drive>` is the drive on which the Web Publishing Engine component of your FileMaker server deployment resides.
  - For Apache (Mac OS): `/Library/WebServer/Documents`
- one of the include_path directories in your PHP installation. The default location for Mac OS X is `/usr/lib/php`.

Where to go from here

Here are some suggestions to get started developing Custom Web Publishing solutions:

- Use FileMaker Server Admin Console to enable Custom Web Publishing. See FileMaker Server Help and *FileMaker Server Getting Started Guide*.
- In FileMaker Pro, open each FileMaker database that you want to publish and make sure the database has the appropriate extended privilege(s) enabled for Custom Web Publishing. See “Enabling Custom Web Publishing with PHP for databases” on page 13.
- To learn how to access data in FileMaker databases using the FileMaker API for PHP, see chapter 5, “Using the FileMaker API for PHP.”

Chapter 3
Preparing databases for Custom Web Publishing

Before you can use Custom Web Publishing with a database, you must prepare the database and protect it from unauthorized access.

Enabling Custom Web Publishing with PHP for databases

You must enable Custom Web Publishing with PHP in each database you want to publish. Otherwise, web users cannot use Custom Web Publishing to access the database, even if it is hosted by a FileMaker Server that is configured to support a Web Publishing Engine.

To enable Custom Web Publishing for a database:

1. In FileMaker Pro, open the database you want to publish using an account with a Full Access or Manage Extended Privileges privilege set.
2. Assign the **fmphp** extended privilege to one or more privilege sets to allow Custom Web Publishing with PHP.
3. Assign the privilege set(s) with the Custom Web Publishing extended privilege to the appropriate accounts (for example, the Admin and Guest accounts).

   Important When defining account names and passwords for Custom Web Publishing solutions, use printable ASCII characters; for example, **a-z**, **A-Z**, and **0-9**. For more secure account names and passwords, include certain non-alphanumeric characters such as an exclamation point (!) or percent sign (%). Colons (:) are not allowed. For details on setting up accounts, see FileMaker Pro Help.

4. Using the FileMaker Server Admin Console, verify that hosting is properly configured for the database, and that it is accessible to the FileMaker Server. See FileMaker Server Help for instructions.

Note Because Custom Web Publishing with PHP does not use persistent database sessions, references to an external ODBC data source in the FileMaker Pro relationships graph may limit the functionality available to your PHP solution. If your database accesses data from an external SQL data source, you may not be able to update the external table's record data.

Creating layouts for Custom Web Publishing with PHP

Custom Web Publishing with PHP does not provide direct table access to data in a FileMaker Pro database, but uses the layouts defined in the databases. While there is no requirement to create a unique layout for Custom Web Publishing with PHP, creating a layout specifically for a PHP solution may be beneficial for several reasons:

- Improve performance by creating a layout that is limited to the fields, labels, calculations, and portals that you need to include in the PHP solution.
- Simplify your PHP code by doing less data processing because the records have fewer fields.
- Separate the interface design work from the data so that you can tailor the interface for the web user.

Protecting your published databases

Custom Web Publishing with PHP enables you to restrict access to your published databases. You can use these methods:

- Require passwords for database accounts used for Custom Web Publishing with PHP.
- Enable the Custom Web Publishing with PHP extended privilege only in those privilege sets for which you want to allow access.
- Disable Custom Web Publishing with PHP for a specific database by deselecting the fmphp extended privilege for all privilege sets in that database. See FileMaker Pro Help.
- Enable or disable Custom Web Publishing for all Custom Web Publishing solutions in the Web Publishing Engine using FileMaker Server Admin Console. See *FileMaker Server Getting Started Guide* and FileMaker Server Help.
- Configure your web server to restrict the IP addresses that can access your databases via the Web Publishing Engine. For example, specify that only web users from the IP address 192.168.100.101 can access your databases. For information on restricting IP addresses, see the documentation for your web server.
- Use Secure Sockets Layer (SSL) encryption for communications between your web server and web browsers. SSL encryption converts information exchanged between servers and clients into unintelligible information using mathematical formulas known as *ciphers*. These ciphers are used to transform the information back into understandable data through encryption keys. For information on enabling and configuring SSL, see the documentation for your web server.

For more information on securing your database, see the *FileMaker Pro User's Guide*, available as a PDF file from http://www.filemaker.com/documentation.

Accessing a protected database

When a web user accesses a database using a PHP solution, the PHP code must provide the credentials to the database using the FileMaker API for PHP. If the Guest account for the database is disabled, or does not have the **fmphp** extended privilege enabled, the FileMaker API for PHP returns an error and your PHP code must provide login information for the user.

The FileMaker API for PHP tutorial includes an example showing how to use the `setProperty()` method to set the username and password for a protected database. See "FileMaker API for PHP Tutorial" on page 24.

The following list summarizes the process that occurs when using Custom Web Publishing to access a database:

- If no password has been assigned for a Custom Web Publishing-enabled account, the PHP solution needs to provide the account name only.
- If the Guest account is disabled, then the PHP solution needs to provide an account name and password. The PHP solution can either prompt the web user for the account name and password, or it can store the account name and password in the PHP code. The account must have the extended privilege **fmphp** enabled.
- If the Guest account is enabled and has the **fmphp** extended privilege enabled:
  - The PHP solution does not need to prompt web users for an account name and password when opening a file. All web users are automatically logged in with the Guest account and assume the Guest account privileges.
  - The default privilege set for Guest accounts provides “read-only” access. You can change the default privileges, including extended privileges, for this account. See FileMaker Pro Help.
- The PHP solution can use the Re-Login script step to allow users to log in using a different account (for example, to switch from the Guest account to an account with more privileges). See FileMaker Pro Help. However, because PHP connections do not use persistent database sessions, the PHP solution must store the account name and password to use them for each subsequent request.

  Note By default, web users cannot change their account passwords from a web browser. You can enable this feature for a database using the Change Password script step, which allows web users to change their passwords from browser. See FileMaker Pro Help.

Publishing the contents of container fields on the web

The contents of a container field can be embedded in the database, linked by reference using a relative path, or stored externally.

Container fields embedded in a database

If a container field stores the actual files in the FileMaker database, follow these steps to use the container field objects in a PHP solution:

- Use FileMaker API for PHP to define the database object (`$fm`) with the appropriate credentials (account name and password).

```
$fm = new FileMaker();
$fm->setProperty('database', $databaseName);
$fm->setProperty('username', $userName);
$fm->setProperty('password', $passWord);
```

- Use the correct HTML tags to indicate the type of web-compatible object that is contained in the container field, and create a URL string that represents the file path for the HTML tag’s source attribute.

```
<IMG src="img.php?-url=<?php echo urlencode($record->getField('Cover
Image')); ?>">
```

- Then use the `getContainerData()` method to retrieve the container field object.

```
echo $fm->getContainerData($_GET['-url']);
```

The FileMaker API for PHP tutorial includes additional examples showing how to use container fields. See “FileMaker API for PHP Tutorial” on page 24.

Note The Web Publishing Engine supports progressive download of audio files (.mp3), video files (.mov, .mp4, and .avi recommended), and PDF files for interactive containers. For example, a web user may start viewing a movie even if the entire movie file has not yet downloaded. To allow for progressive download, you may need to create the files using options that support streaming or that optimize for display on the web. For example, create PDF files using the “Optimize for Web Viewing” option.

Container fields with referenced files

If a container field stores a file reference, you can use the `getContainerData()` method retrieve the container field objects from the database in your PHP code, or you can use the `getContainerDataURL()` method to retrieve a fully qualified URL for the container field object.

You must also follow these steps to publish the referenced files using the Web Publishing Engine:

1. Store the container object files in the Web folder inside the FileMaker Pro folder.
2. In FileMaker Pro, insert the objects into the container field and select the **Store only a reference to the file** option.
3. Copy or move the referenced object files in the Web folder to the same relative path location in the root folder of the web server software.
   - For IIS (Windows): `<drive>:\Inetpub\wwwroot` where `<drive>` is the drive on which the Web Publishing Engine component of your FileMaker server deployment resides.
   - For Apache (Mac OS): `/Library/WebServer/Documents`

Notes

- For container objects stored as file references, your web server must be configured to support the MIME (Multipurpose Internet Mail Extensions) types for the kinds of files you want to serve, such as movies. Your web server determines the support for the current MIME types registered for the Internet. The Web Publishing Engine does not change a web server’s support for MIME. For more information, see the documentation for your web server.
- All QuickTime movies stored in a container field are stored by reference.

Container fields with externally stored data

If a container field stores objects externally — that is, if you selected **Store container data externally** in the Field Options dialog — your PHP code needs to use the `getContainerDataURL()` method to retrieve a fully qualified URL for the container field object.

Use FileMaker API for PHP to define the database object with the appropriate credentials (account name and password), and then use the `getContainerDataURL()` method.

Example showing images using HTML img tag

```
$fm=new FileMaker($database, $hostspec, $user, $password);
$findCommand = $fm->newFindCommand($layout);
$findCommand->addFindCriterion('type', 'png');
$result = $findCommand->execute();
$records = $result->getRecords();
foreach ($records as $record) {
        echo $record->getField('container').'<br>';
        // For images, use the HTML img tag
        echo '<img src="'.$fm->
          getContainerDataURL($record->getField('container')) .'">';
        break;
        }
```

Example showing embedded data using HTML embed tag

```
$fm=new FileMaker($database, $hostspec, $user, $password);
$findCommand = $fm->newFindCommand($layout);
$findCommand->addFindCriterion('type', 'pdf');
$result = $findCommand->execute();
$records = $result->getRecords();
foreach ($records as $record) {
        echo $record->getField('container').'<br>';
        // For movies and PDF files, use the HTML embed tag
        //echo '<embed src="'.$fm->
          getContainerDataURL($record->getField('container')) .'">';
        break;
        }
```

If a container field stores objects externally, then use the Upload Database assistant to transfer database files from the client file system to FileMaker Server. The Upload Database assistant transfers the database and the container field objects to the correct folders on your server for hosting. See FileMaker Server Help for more information on how to use the Upload Database assistant. See FileMaker Pro Help for information on setting up container fields to store data externally.

If you manually upload a database that uses a container field with externally stored objects, then you must follow these steps to publish the externally stored container objects using the Web Publishing Engine.

To upload a database manually:

1. Place database file in the proper location on the server. Place the FileMaker Pro database files that you want FileMaker Server to open — or shortcuts (Windows) or aliases (Mac OS) to those files — in the following folders:
   - Windows (32-bit): `[drive]:\Program Files\FileMaker\FileMaker Server\Data\Databases\`
   - Windows (64-bit): `[drive]:\Program Files (x86)\FileMaker\FileMaker Server\Data\Databases\`
   - Mac OS: `/Library/FileMaker Server/Data/Databases/`

   Or you can place the files in an optionally specified additional database folder.
2. In the folder where you placed the database, create a folder named RC_Data_FMS, if it doesn't already exist.
3. In the RC_Data_FMS folder, create a folder with a name that matches the name of your database. For example, if your database is named Customers, then create a folder named Customers. Place the externally stored objects in the new folder you created.

   Note When databases are hosted on FileMaker Server, there is no way for multiple databases to share a common folder of container objects. The container objects for each database needs to be in a folder identified by that database's name.
4. For files that will be shared from Mac OS, change the files to belong to the **fmsadmin** group.

   For more information about manually uploading databases, see FileMaker Server Help.

Note The Web Publishing Engine supports progressive download of audio files (.mp3), video files (.mov, .mp4, and .avi recommended), and PDF files for interactive containers. For example, a web user may start viewing a movie even if the entire movie file has not yet downloaded. To allow for progressive download, you may need to create the files using options that support streaming or that optimize for display on the web. For example, create PDF files using the "Optimize for Web Viewing" option.

How web users view container field objects

When you publish a database using the Web Publishing Engine, the following limitations apply to container field objects:

- Web users cannot modify or add to the contents of container fields. Web users cannot use container fields to upload objects to the database.
- For databases that use a container field with thumbnails enabled, the Web Publishing Engine downloads the full file, not a thumbnail.

FileMaker scripts and Custom Web Publishing

The Manage Scripts feature in FileMaker Pro can automate frequently performed tasks, or combine several tasks. When used with Custom Web Publishing, FileMaker scripts allow web users to perform a series of tasks. FileMaker scripts also allow tasks that are not supported in any other way, such as using the Change Password script step to allow web users to change passwords from a browser.

FileMaker supports over 65 script steps in Custom Web Publishing. To see script steps that are not supported, select **Custom Web Publishing** from the **Show Compatibility** list in the Edit Script window in FileMaker Pro. Dimmed script steps are not supported for Custom Web Publishing. For information on creating scripts, see FileMaker Pro Help.

Script tips and considerations

Although many script steps work identically on the web, there are several that work differently. See “Script behavior in Custom Web Publishing solutions” on page 20. Before sharing your database, evaluate all scripts that will be executed from a web browser. Be sure to log in with different user accounts to make sure they work as expected for all clients.

Keep these tips and considerations in mind:

- Use accounts and privileges to restrict the set of scripts that a web user can execute. Verify that the scripts contain only web-compatible script steps, and only provide access to scripts that should be used from a web browser.
- Consider the side effects of scripts that execute a combination of steps that are controlled by access privileges. For example, if a script includes a step to delete records, and a web user does not log in with an account that allows record deletion, the script will not execute the Delete Records script step. However, the script might continue to run, which could lead to unexpected results.
- In the Edit Script window, select **Run script with full access privileges** to allow scripts to perform tasks for which you would not grant access by an individual. For example, you can prevent users from deleting records by restricting their accounts and privileges, but still allow users to run a script that would delete certain types of records under conditions predefined within the script.
- If your scripts contain steps that are unsupported—for example, steps that are not web-compatible—use the **Allow User Abort** script step to determine how subsequent steps are handled:
  - If the **Allow User Abort** script step option is enabled (on), unsupported script steps stop the script from continuing.
  - If **Allow User Abort** is off, unsupported script steps are skipped and the script continues to execute.
  - If this script step is not included, scripts are executed as if the feature is enabled, so unsupported script steps stop scripts.
- Some scripts that work with one step from a FileMaker Pro client may require an additional Commit Record/Request step to save the data to the host. Because web users don’t have a direct connection to the host, they aren’t notified when data changes. For example, features like conditional value lists aren’t as responsive for web users because the data must be saved to the host before the effects are seen in the value list field.
- Any script that modifies data should include the Commit Record/Request step, because data changes won’t be visible in the browser until the data is saved or “submitted” to the server. This includes several script steps like Cut, Copy, Paste, and so on. Many single-step actions should be converted into scripts to include the Commit Record/Request step. When designing scripts that will be executed from a web browser, include the Commit Record/Request step at the end of a script to make sure all changes are saved.

- To create conditional scripts based on the type of client, use the Get(ApplicationVersion) function. If the value returned includes “Web Publishing Engine” you know that the current user is accessing your database with Custom Web Publishing. For more information on functions, see FileMaker Pro Help.
- After converting your files, you should open each script that web users might run and select **Web Publishing** from the Show Compatibility list in the Edit Script window to verify that the script will execute properly with Instant Web Publishing.

Script behavior in Custom Web Publishing solutions

The following script steps function differently on the web than in FileMaker Pro. For information on all script steps, see FileMaker Pro Help.

Script step	Behavior in Custom Web Publishing solutions
Perform Script	Scripts cannot perform in other files, unless the files are hosted on FileMaker Server and Custom Web Publishing is enabled in the other files.
Exit Application	Logs off web users, closes all windows, but does not exit the web browser application.
Allow User Abort	Determines how unsupported script steps are handled. Enable to stop scripts from continuing, and disable to skip unsupported steps. See “Script tips and considerations” on page 19 for more details. Note Web users cannot abort Custom Web Publishing scripts, but this option allows unsupported script steps to stop the script from continuing.
Set Error Capture	This is always enabled with Custom Web Publishing. Web users cannot abort Custom Web Publishing scripts.
Pause/Resume script	Although these script steps are supported in Custom Web Publishing, you should avoid using them. When a Pause step is executed, the script pauses. Only a script containing the Resume script step can make it resume execution. If the script remains in a paused state until the session times out, then the script will not be completed.
Sort Records	You must save a sort order with the Sort Records script step to execute in Custom Web Publishing.
Open URL	This script step has no effect in a Custom Web Publishing solution.
Go to Field	You cannot use Go to Field to make a particular field active in the web browser, but you can use this script step in conjunction with other script steps to perform tasks. For example, you can go to a field, copy the contents, go to another field and paste the value. To see the effect in the browser, be sure to save the record with the Commit Record script step.
Commit Record/Request	Submits the record to the database.

Script triggers and Custom Web Publishing solutions

In FileMaker Pro, both scripts and user actions (such as the user clicking a field) can activate script triggers. But in Custom Web Publishing, only scripts can activate script triggers. For example, if a Custom Web Publishing user clicks a field that has an OnObjectEnter script trigger, the trigger is not activated. However, if a script causes the focus to move to the field, then the OnObjectEnter script trigger is activated. For more information on script triggers, see FileMaker Pro Help.

Note For FileMaker Pro 12, the File Options dialog box has changed. As a result, to specify that you want a script performed when a file is opened, you need to use the OnFirstWindowOpen script trigger. Similarly, to specify that you want a script performed when a file is closed, you need to use the OnLastWindowClose script trigger.

Chapter 4
Overview of Custom Web Publishing with PHP

The FileMaker API for PHP helps you integrate data from FileMaker Pro databases into PHP solutions. This chapter describes how PHP works with the FileMaker Server Custom Web Publishing Engine. For more detailed information about the FileMaker API for PHP, see chapter 5, “Using the FileMaker API for PHP.”

How the Web Publishing Engine works with PHP solutions

FileMaker Server is composed of three components: a web server, the Web Publishing Engine, and the Database Server. (These components may be deployed on one machine, two machines, or three machines. See *FileMaker Server Getting Started Guide* for information.) FileMaker Server hosts the PHP solution when you place the PHP files on the web server where the PHP engine is installed.

- When a web user opens a PHP solution, the web server routes the request to the PHP engine, which processes the PHP code.
- If the PHP code contains calls to the FileMaker API for PHP, those calls are interpreted and sent as requests to the Web Publishing Engine.
- The Web Publishing Engine requests data from databases that are hosted on the Database Server.
- The Database Server sends the requested data to the Web Publishing Engine.
- The Web Publishing Engine sends the data to the PHP engine on the web server in response to the API call.
- The PHP solution processes the data, and displays it for the web user.

General steps for Custom Web Publishing with PHP

Here is a summary of the steps for using Custom Web Publishing with PHP:

1. In the Admin Console, make sure **Enable PHP publishing** is selected. See *FileMaker Server Getting Started Guide*.

2. In the Admin Console, choose the **Databases** pane and make sure each FileMaker database that you’re publishing has the **fmphp** extended privilege enabled for Custom Web Publishing with PHP.

   If necessary, use FileMaker Pro to enable Custom Web Publishing for a database. See chapter 3, “Preparing databases for Custom Web Publishing.”

   Note Make sure that you use equivalent FileMaker database privilege sets when developing PHP solutions that will be given to the end user. Otherwise, you may have access to layouts and features in the FileMaker database that will not be available to the end user, causing inconsistent behavior.

3. Use PHP authoring tools to create your PHP solution, incorporating the FileMaker API functions into your PHP code to access your FileMaker data. See chapter 5, “Using the FileMaker API for PHP.”

4. Copy or move your site directory structure and files to the web server root folder.
   - For IIS (Windows): `<drive>:\Inetpub\wwwroot` where `<drive>` is the drive on which the Web Publishing Engine component of your FileMaker server deployment resides.
   - For Apache (Mac OS): `/Library/WebServer/Documents`

5. If a database container field stores a file reference instead of an actual file, the referenced container object must be stored in the FileMaker Pro Web folder when the record is created or edited. You must copy or move the object to a folder with the same relative location in the root folder of the web server software.

   See “Publishing the contents of container fields on the web” on page 15.

6. Make sure that security mechanisms for your site or program are in place.

7. Test your site using the same accounts and privileges defined for web users.

8. Make the site available and known to users. The URL that the web user enters follows this format:

   ```
   http://<server>/<site_path>
   ```

   - `<server>` is the machine on which the FileMaker Server resides
   - `<site_path>` is the relative path to the home page for your site, determined by the directory structure you used in step 4 above.

   For example, if your web server is 192.168.123.101 and your site home page is on the web server at c:\Inetpub\wwwroot\customers\index.php, then the web user would enter this URL:

   ```
   http://192.168.123.101/customers/index.php
   ```

Note PHP 5 uses Latin-1 (ISO-8859-1) encoding. FileMaker Server returns Unicode (UTF-8) data. Use the FileMaker Server Admin Console to specify the default character encoding for your site. For PHP sites, you can specify either UTF-8 or ISO-8859-1; UTF-8 is recommended. Specify the same setting for the `charset` attribute in the `<HEAD>` section of your site PHP files.

For information on deploying and using a PHP solution, see chapter 6, “Staging, testing, and monitoring a site.”

Chapter 5
Using the FileMaker API for PHP

The FileMaker API for PHP implements a PHP class—the FileMaker class—that provides an object-oriented interface to FileMaker databases. The FileMaker API for PHP enables both data and logic stored in FileMaker Pro databases to be accessed and published on the web, or exported to other applications.

The FileMaker API for PHP allows PHP code to perform the same kind of functions you already have available in FileMaker Pro databases:

- create, delete, edit, and duplicate records
- perform find requests
- perform field and record validation
- use layouts
- run FileMaker scripts
- display portals and related records
- use value lists

This chapter describes how to use the FileMaker class objects and methods to add these common functions to a PHP solution. This chapter does not cover the entire the FileMaker API for PHP, but introduces key objects and methods.

Where to get additional information

To learn more about the FileMaker API for PHP, see the following resources.

If you already have a PHP engine installed and configured and you want to add only the FileMaker API for PHP, see “Manually installing the FileMaker API for PHP” on page 11.

FileMaker API for PHP Reference

If you installed the FileMaker API for PHP, you can find reference information on the web server component of your FileMaker Server deployment.

- For IIS (Windows):
  `<drive>:\Program Files\FileMaker\FileMaker Server\Documentation\PHP API Documentation\index.html`
  where `<drive>` is the drive on which the web server component of your FileMaker server deployment resides.
- For Apache (Mac OS): `/Library/FileMaker Server/Documentation/PHP API Documentation/index.html`

FileMaker API for PHP Tutorial

If you installed the FileMaker API for PHP, you can find a tutorial on the web server component of your FileMaker Server deployment.

- For IIS (Windows): `<drive>:\Program Files\FileMaker\FileMaker Server\Examples\PHP\Tutorial`
  where `<drive>` is the drive on which the web server component of your FileMaker server deployment resides.
- For Apache (Mac OS): `/Library/FileMaker Server/Examples/PHP/Tutorial`

To host these PHP tutorial files, copy them to the web server root folder.

FileMaker API for PHP Examples

If you installed the FileMaker API for PHP, you can find additional examples on the web server component of your FileMaker Server deployment.

- For IIS (Windows): `<drive>:\Program Files\FileMaker\FileMaker Server\Examples\PHP\API Examples` where `<drive>` is the drive on which the web server component of your FileMaker server deployment resides.
- For Apache (Mac OS): `/Library/FileMaker Server/Examples/PHP/API Examples`

To host these API example files, copy them to the web server root folder.

Using the FileMaker class

To use the FileMaker class in your PHP solution, add the following statement to your PHP code:

```
require_once ('FileMaker.php');
```

FileMaker class objects

The FileMaker class defines class objects that you can use to retrieve data from FileMaker Pro databases.

Class Object	Use the object to
FileMaker database	Define the database properties Connect to a FileMaker Pro database Get information about the FileMaker API for PHP
Command	Create commands that add records, delete records, duplicate records, edit records, perform find requests, and perform scripts.
Layout	Work with database layouts
Record	Work with record data
Field	Work with field data
Related set	Work with portal records
Result	Process the records returned from a Find request
Error	Check whether an error has occurred Process any errors

FileMaker command objects

The FileMaker class defines a base command object that you use to instantiate a specific command and to specify the command's parameters. To execute the command, you must call the `execute()` method.

The FileMaker class defines the following specific commands:

- Add command
- Compound Find command
- Delete command
- Duplicate command
- Edit command
- Find command, Find All command, Find Any command
- Find Request command, which gets added to a Compound Find command
- Perform Script command

These commands are described in more detail in the following sections:

- "Working with records" on page 26
- "Running FileMaker scripts" on page 27
- "Performing find requests" on page 33

Connecting to a FileMaker database

The FileMaker class defines a database object that you instantiate in order to connect to a server or to a database. Define the object properties with the class constructor, or by calling the `setProperty()` method.

Example: Connecting to a server to get a list of databases

```
$fm = new FileMaker();
$databases = $fm->listDatabases();
```

Example: Connecting to a specific database on a server

The username and password properties determine the privilege set for this connection.

```
$fm = new FileMaker();
$fm->setProperty('database', 'questionnaire');
$fm->setProperty('hostspec', 'http://192.168.100.110');
$fm->setProperty('username', 'web');
$fm->setProperty('password', 'web');
```

Note The hostspec property defaults to the value `http://localhost`. If the PHP engine is running on the same machine as the web server component of the FileMaker Server deployment, there is no need to specify the hostspec property. If the PHP engine is on a different machine, use the hostspec property to specify the location of the web server component of the FileMaker Server deployment.

Working with records

The FileMaker class defines a record object that you instantiate to work with records. An instance of a record object represents one record from a FileMaker Pro database. Use a record object with Add, Delete, Duplicate, and Edit commands to change the data in the record. The Find commands—Find, Find All, Find Any, and Compound Find—return an array of record objects.

Creating a record

There are two ways to create a record:

- Use the `createRecord()` method, specifying a layout name, and optionally specifying an array of field values. You can also set values individually in the new record object.

  The `createRecord()` method does not save the new record to the database. To save the record to the database, call the `commit()` method.

  For example:

  ```
  $rec = $fm->createRecord('Form View', $values);
  $result = $rec->commit();
  ```

- Use the Add command. Use the `newAddCommand()` method to create a FileMaker_Command_Add object, specifying the layout name and an array with the record data. To save the record to the database, call the `execute()` method.

  For example:

  ```
  $newAdd = $fm->newAddCommand('Respondent', $respondent_data);
  $result = $newAdd->execute();
  ```

Duplicating a record

Duplicate an existing record using the Duplicate command. Use the `newDuplicateCommand()` method to create a FileMaker_Command_Duplicate object, specifying the layout name and the record ID of the record that you want to duplicate. Then, duplicate the record by calling the `execute()` method.

Example

```
$newDuplicate = $fm->newDuplicateCommand('Respondent', $rec_ID);
$result = $newDuplicate->execute();
```

Editing a record

There are two ways to edit a record:

- Using the Edit command. Use the `newEditCommand()` method to create a FileMaker_Command_Edit object, specifying the layout name, the record ID of the record you want to edit, and an array of values that you want to update. Then, edit the record by calling the `execute()` method.

  For example:

  ```
  $newEdit = $fm->newEditCommand('Respondent', $rec_ID, $respondent_data);
  $result = $newEdit->execute();
  ```

- Using a record object. Retrieve a record from the database, change field values, and then edit the record by calling the `commit()` method.

  For example:

```
$rec = $fm->getRecordById('Form View', $rec_ID);
$rec->setField('Name', $nameEntered);
$result = $rec->commit();
```

Deleting a record

There are two ways to delete a record:

- Retrieve a record from the database, and then call the `delete()` method.

  For example:

```
$rec = $fm->getRecordById('Form View', $rec_ID);
$rec->delete();
```

- Delete an existing record using the Delete command. Use the `newDeleteCommand()` method to create a FileMaker_Command_Delete object, specifying the layout name and the record ID of the record you want to delete. Then, delete the record by calling the `execute()` method.

  For example:

```
$newDelete = $fm->newDeleteCommand('Respondent', $rec_ID);
$result = $newDelete->execute();
```

Running FileMaker scripts

A FileMaker script is a named set of script steps. The FileMaker class defines several methods that allow you to work with FileMaker scripts defined in a FileMaker Pro database. For information on web-compatible script steps (the script steps that can be performed in a web solution), see “FileMaker scripts and Custom Web Publishing” on page 18.

Obtaining the list of available scripts

Use the `listScripts()` method to get a list of available scripts from the currently connected database. The `listScripts()` method returns an array of scripts that can be executed by the username and password specified when the database connection was defined. (See “Connecting to a FileMaker database” on page 25.)

Example

```
$scripts = $fm->listScripts();
```

Running a FileMaker script

Use the `newPerformScriptCommand()` method to create a FileMaker_Command_PerformScript object, specifying the layout, script name, and any script parameters. Then, perform the script by calling the `execute()` method.

Example

```
$newPerformScript = $fm->newPerformScriptCommand('Order Summary',
'ComputeTotal');
$result = $newPerformScript->execute();
```

Running a script before executing a command

Use the `setPreCommandScript()` method to specify a script that runs before a command is run. The following example uses a Find command, but you can use the `setPreCommandScript()` method with any command.

Example

```
$findCommand = $fm->newFindCommand('Students');
$findCommand->addFindCriterion('GPA', $searchValue);
$findCommand->setPreCommandScript('UpdateGPA');
$result = $findCommand->execute();
```

Running a script before sorting a result set

Use the `setPreSortScript()` method to specify a script that is run after a Find result set is generated, but before the result set is sorted. For more information, see “Using the Find command” on page 34.

Example

```
$findCommand = $fm->newFindCommand('Students');
$findCommand->setPreSortScript('RemoveExpelled');
```

Running a script after the result set is generated

Use the `setScript()` method to specify a script that is run after a Find result set is generated. For more information, see “Using the Find command” on page 34.

Example

```
$findCommand = $fm->newFindCommand('Students');
$findCommand->setScript('myScript','param1|param2|param3');
```

Script execution order

You can specify the `setPreCommandScript()`, `setPreSortScript()`, and `setScript()` methods in conjunction with the `setResultLayout()` and `addSortRule()` methods for a single command. Here is the order in which FileMaker Server and the Web Publishing Engine process these methods:

1. Run the script specified on the `setPreCommandScript()` method, if specified.

2. Process the command itself, such as a Find or Delete Record command.

3. Run the script specified on the `setPreSortScript()` method, if specified.

4. Sort the Find result set, if the `addSortRule()` method was specified.

5. Process the `setResultLayout()` method to switch to a different layout, if this is specified.

6. Run the script specified on the `setScript()` method, if specified.

7. Return the final Find result set.

If one of the above steps generates an error code, the command execution stops; any steps that follow are not executed. However, any prior steps in the request are still executed.

For example, consider a command that deletes the current record, sorts the records, and then executes a script. If the `addSortRule()` method specifies a non-existent field, the request deletes the current record and returns error code 102 (“Field is missing”), but does not execute the script.

Working with FileMaker layouts

A layout is the arrangement of fields, objects, pictures, and layout parts that represents the way information is organized and presented when the user browses, previews, or prints records. The FileMaker class defines several methods that allow you to work with the layouts defined in a FileMaker Pro database. You can get information about layouts from several of the FileMaker class objects.

With this class object	Use these methods
Database	▪ `listLayouts()` obtains a list of available layout names. ▪ `getLayout()` obtains a layout object by specifying a layout name.
Layout	▪ `getName()` retrieves the layout name of a specific layout object. ▪ `listFields()` retrieves an array of all field names used in a layout. ▪ `getFields()` retrieves an associative array with the names of all fields as keys, and the associated FileMaker_Field objects as array values. ▪ `listValueLists()` retrieves an array of value list names. ▪ `listRelatedSets()` retrieves an array of related sets names. ▪ `getDatabase()` returns the name of the database.
Record	▪ `getLayout()` returns the layout object associated with a specific record.
Field	▪ `getLayout()` returns the layout object containing specific field.
Command	▪ `setResultLayout()` returns the command’s results in a layout different from the current layout.

Using portals

A portal is table that displays rows of data from one or more related records. The FileMaker class defines a related set object and several methods that allow you to work with portals defined in a FileMaker Pro database.

A related set object is an array of record objects from the related portal; each record object represents one row of data in the portal.

Listing the portals defined on a specific layout

For a specific layout object, use the `listRelatedSets()` method to retrieve a list of table names for all portals defined in this layout.

Example

```
$tableNames = $currentLayout->listRelatedSets();
```

Obtaining portal names for a specific result object

For a specific FileMaker_Result object, use the `getRelatedSets()` method to retrieve the names of all portals in this record.

Example

```
$relatedSetsNames = $result->getRelatedSets();
```

Obtaining information about portals for a specific layout

For a specific layout object, use the `getRelatedSets()` method to retrieve an array of FileMaker_RelatedSet objects that describe the portals in the layout. The returned array is an associative array with the table names as the array keys, and the associated FileMaker_RelatedSet objects as the array values.

Example

```
$relatedSetsArray = $currentLayout->getRelatedSets();
```

Obtaining information for a specific portal

For a specific layout object, use the `getRelatedSet()` method to retrieve the FileMaker_RelatedSet object that describes a specific portal.

Example

```
$relatedSet = $currentLayout->getRelatedSet('customers');
```

Obtaining the table name for a portal

For a related set object, use the `getName()` method to get the table name for the portal.

Example

```
$tableName = $relatedSet->getName();
```

Obtaining the portal records for a specific record

For a specific record object, use the `getRelatedSet()` method to retrieve an array of related records for a specific portal on that record.

Example

```
$relatedRecordsArray = $currentRecord->getRelatedSet('customers');
```

Creating a new record in a portal

Use the `newRelatedRecord()` method to create a new record in the specified related set, and commit the change to the database by calling the `commit()` method.

Example

```
//create a new portal row in the 'customer' portal
$new_row = $currentRecord->newRelatedRecord('customer');

//set the field values in the new portal row
$new_row->setField('customer::name', $newName);
$new_row->setField('customer::company', $newCompany);

$result = $new_row->commit();
```

Deleting a record from a portal

Use the `delete()` method to delete a record in a portal.

Example

```
$relatedSet = $currentRecord->getRelatedSet('customers');
/* Runs through each of the portal rows */
foreach ($relatedSet as $nextRow) {

    $nameField = $nextRow->getField('customer::name')
    if ($nameField == $badName ) {
        $result =   $newRow->delete();
    }
}
```

Using value lists

A value list is set of predefined choices. The FileMaker class defines several methods that allow you to work with value lists defined in a FileMaker Pro database.

Obtaining the names of all value lists for a specific layout

For a specific layout object, use the `listValueLists()` method to retrieve an array that contains value list names.

Example

```
$valueListNames = $currentLayout->listValueLists();
```

Obtaining an array of all value lists for a specific layout

For a specific layout object, use the `getValueListsTwoFields()` method to retrieve an array containing the values from all value lists. The returned array is an associative array. The array keys are the value list names, and the array values are associative arrays that list the display names and their corresponding choices from each value list.

Example

```
$valueListsArray = $currentLayout->getValueListsTwoFields();
```

Note Although the `getValueLists()` method is still supported in the FileMaker API for PHP, it will be deprecated. Instead, use the `getValueListsTwoFields()` method.

Obtaining the values for a named value list

For a specific layout object, use the `getValueListTwoFields()` method to get an array of choices defined for the named value list. The returned array is an associative array that contains the displayed values from the second field of the value list as the keys, and the associated stored values from the first field as the array values.

Depending on the options selected in the **Specify Fields for Value List** dialog box in the FileMaker database, the `getValueListTwoFields()` method returns the value in the first field only, the value in the second field only, or the values in both fields of a value list as the stored and displayed values.

- If **Also display values from second field** is not selected, the `getValueListTwoFields()` method returns the value from the first field of the value list as both the stored value and the displayed value.
- If **Also display values from second field** and **Show values only from second field** are both selected, the `getValueListTwoFields()` method returns the value from the first field as the stored value, and the value from the second field as the displayed value.
- If **Also display values from second field** is selected and **Show values only from second field** is not selected, the `getValueListTwoFields()` method returns the value from the first field as the stored value, and both values from the first and second fields as the displayed value.

Use an iterator with the `getValueListTwoFields()` method to find the displayed value and stored value.

Example

```
$layout = $fm->getLayout('customers');
$valuearray = $layout->getValueListTwoFields("region", 4);
foreach ($valuearray as $displayValue => $value) {
    ....
}
```

Notes

- Although the `getValueList()` method is still supported in the FileMaker API for PHP, it will be deprecated. Instead, use the `getValueListTwoFields()` method.
- When using the `getValueListTwoFields()` method, be sure to use a `foreach` loop to loop through the associative array. Do not use a `for` loop because it can return unexpected results.

Performing find requests

The FileMaker class defines four kinds of Find command objects:

- Find All command. See “Using the Find All command” on page 33.
- Find Any command. See “Using the Find Any command” on page 34.
- Find command. See “Using the Find command” on page 34.
- Compound Find command. See “Using a Compound Find command” on page 34.

The FileMaker class also defines several methods that can be used for all four types of Find commands:

- Use the `addSortRule()` method to add a rule defining how the result set is sorted. Use the `clearSortRules()` method to clear all sort rules that have been defined.
- Use the `setLogicalOperator()` method to change between logical AND searches and logical OR searches.
- Use the `setRange()` method to request only part of the result set. Use the `getRange()` method to retrieve the current range definition.

  Using the `setRange()` method can improve the performance of your solution by reducing the number records that are returned by the Find request. For example, if a Find request returns 100 records, you can split the result set into five groups of 20 records each rather than processing all 100 records at once.
- You can execute FileMaker scripts in conjunction with Find commands.
  - To run a script before executing the Find command, use the `setPreCommandScript()` method.
  - To run a script before sorting the result set, use the `setPreSortScript()` method.
  - To run a script after a result set is generated, but before the result set is sorted, use the `setScript()` method.

Using the Find All command

Use the Find All command to retrieve all records from a specified layout. Use the `newFindAllCommand()` method, specifying a specific layout, to create a FileMaker_Command_FindAll object. Then, perform the find request by calling the `execute()` method.

Example

```
$findCommand = $fm->newFindAllCommand('Form View');
$result = $findCommand->execute;
```

Note When using the Find All command, avoid computer memory overload problems by specifying a default maximum number of records to return per page.

Using the Find Any command

Use the Find Any command to retrieve one random record from a specified layout. Use the `newFindAnyCommand()` method, specifying a specific layout, to create a FileMaker_Command_FindAny object. Then, perform the find request by calling the `execute()` method.

Example

```
$findCommand = $fm->newFindAnyCommand('Form View');
$result = $findCommand->execute;
```

Using the Find command

Use the `newFindCommand()` method, specifying a specific layout, to create a FileMaker_Command_Find object. Then, perform the find request by calling the `execute()` method.

Use the `addFindCriterion()` method to add criteria to the find request. Use the `clearFindCriteria()` method to clear all find criteria that have been defined.

Example - Finding a record by field name

```
$findCommand = $fm->newFindCommand('Form View');
$findCommand->addFindCriterion('Questionnaire ID', $active_questionnaire_id);
$result = $findCommand->execute();
```

Example - Adding a sort order

```
$findCommand = $fm->newFindCommand('Customer List');
$findCommand->addSortRule('Title', 1, FILEMAKER_SORT_ASCEND);
$result = $findCommand->execute();
```

Using a Compound Find command

The Compound Find command lets you combine multiple Find Request objects into one command.

To create a Compound Find command:

- Create a FileMaker_Command_CompoundFind object by calling the `newCompoundFindCommand()` method.
- Create one or more FileMaker_Command_FindRequest objects by calling the `newFindRequest()` method.
- Use the `setOmit()` method to indicate records in the result set of a specific Find Request that are to be omitted from the final result set.
- Use the `add()` method to add the Find Request objects to the Compound Find command object.
- Perform the Compound Find command by calling the `execute()` method.

Example - Compound Find command

```
// Create the Compound Find command object
$compoundFind = $fm->newCompoundFindCommand('Form View');

// Create first find request
$findreq1 = $fm->newFindRequest('Form View');

// Create second find request
$findreq2 = $fm->newFindRequest('Form View');

// Create third find request
$findreq3 = $fm->newFindRequest('Form View');

// Specify search criterion for first find request
$findreq1->addFindCriterion('Quantity in Stock', '<100');

// Specify search criterion for second find request
$findreq2->addFindCriterion('Quantity in Stock', '0');
$findreq2->setOmit(true);

// Specify search criterion for third find request
$findreq3->addFindCriterion('Cover Photo Credit', 'The London Morning News');
$findreq3->setOmit(true);

// Add find requests to compound find command
$compoundFind->add(1,$findreq1);
$compoundFind->add(2,$findreq2);
$compoundFind->add(3,$findreq3);

// Set sort order
$compoundFind->addSortRule('Title', 1, FILEMAKER_SORT_DESCEND);

// Execute compound find command
$result = $compoundFind->execute();

// Get records from found set
$records = $result->getRecords();

// Print number of records found
echo 'Found '. count($records) . " results.<br><br>";
```

Processing the records in a result set

- Retrieve an array containing each record in the result set by calling the `getRecords()` method. Each member of the array is a FileMaker_Record object, or an instance of the class name set in the API for instantiating records. The array may be empty if the result set contains no records.
- Get a list of field names for all fields in the result set by calling the `getFields()` method. The method returns only the field names. If you need additional information about the fields, use the associated layout object.
- Get the number of records in the entire found set by calling the `getFoundSetCount()` method.
- Get the number of records in the filtered found set by calling the `getFetchCount()` method. If no range parameters were specified on the Find command, then this value is equal to the result of the `getFoundSetCount()` method. It is always equal to the value of `count($response->getRecords())`.
- For a specific record, use the `getField()` method to return the contents of a field as a string.
- For a specific record, use the `getFieldAsTimestamp()` method to return the contents of a field as a Unix timestamp (the PHP internal representation of a date).
  - If the field is a date field, the timestamp is for the field date at midnight.
  - If the field is a time field, the timestamp is for that time on January 1, 1970.
  - If the field is a timestamp field, the FileMaker timestamp value maps directly to the Unix timestamp.
  - If the specified field is not a date or time field, or if the timestamp generated would be out of range, the `getFieldAsTimestamp()` method return a FileMaker_Error object.
- For a specific record, use the `getContainerData()` method to return a container field object as binary data:

```
<IMG src=”img.php?-url=<?php echo urlencode($record->getField('Cover
Image')); ?>”>
echo $fm->getContainerData($_GET['-url']);
```

- For a specific record, use the `getContainerDataURL()` method to return a fully qualified URL for the container field object:

```
// For images, use the HTML img tag
        echo '<img src="'.$fm->
getContainerDataURL($record->getField('container')) .'">';
// For movies and PDF files, use the HTML embed tag
        //echo '<embed src="'.$fm->
getContainerDataURL($record->getField('container')) .'">';
```

Filtering portal rows returned by find requests

In a solution that has many related records, querying and sorting portal records can be time consuming. To restrict the number of records to display in a related set, use the `setRelatedSetsFilters()` method with find requests. The `setRelatedSetsFilters()` method takes two arguments:

- a related sets filter value: `layout` or `none`.
  - If you specify the value `none`, the Web Publishing Engine returns all rows in the portal, and portal records are not presorted.
  - If you specify the value `layout`, then the settings specified in the FileMaker Pro Portal Setup dialog are respected. The records are sorted based on the sort defined in the Portal Setup dialog, with the record set filtered to start with the initial row specified.
- the maximum number of portal records returned: an integer value or `all`.
  - This value is used only if the Show Vertical Scroll Bar setting is enabled in the Portal Setup dialog. If you specify an integer value, that number of rows after the initial row are returned. If you specify `all`, the Web Publishing Engine returns all of the related records.
  - If the Show Vertical Scroll Bar setting is disabled, the Portal Setup dialog's Number of rows setting determines the maximum number of related records that are returned.

Pre-validating commands, records, and fields

The FileMaker class lets you *pre-validate* field data in a PHP solution on the web server before committing the data to the database.

When deciding whether to use pre-validation, consider the number of data values that the web user is entering. If the user is updating a small number of fields, then you could improve performance by not using pre-validation. But if the user is entering data for many fields, then pre-validation can keep the user from being frustrated by having a record rejected by the database for validation errors.

With the FileMaker class, the PHP engine pre-validates the following field constraints:

- not empty

  Valid data is a non-empty character string. The data must contain at least one character.
- numeric only

  Valid data contains numeric characters only.
- maximum number of characters

  Valid data contains at most the maximum number of characters specified.
- four-digit year

  Valid data is a character string representing a date with a four-digit year in the format M/D/YYYY, where M is a number between 1 and 12 inclusive, D is a number between 1 and 31 inclusive, and YYYY is a four-digit number between 0001 and 4000 inclusive. For example, `1/30/3030` is a valid four-digit year value. However, `4/31/2012` is an invalid four-digit year value because April does not have 31 days. Date validation supports forward slash (/), back slash (\), and hyphen (-) as delimiters. However, the string cannot contain a mix of delimiters. For example, `1\30-2012` is invalid.

- time of day

  Valid data is a character string representing a 12-hour time value in the one of these formats:

  - H
  - H:M
  - H:M:S
  - H:M:S AM/PM
  - H:M AM/PM

  where H is a number between 1 and 12 inclusive; M and S are numbers between 1 and 60 inclusive.

The PHP engine pre-validation supports implicit checking of field data based on the type of the field:

- date

  A field defined as a date field is validated according to the rules of “four-digit year” validation, except the year value can contain 0-4 digits (the year value can be empty). For example, `1/30` is a valid date even though it has no year specified.

- time

  A field defined as a time field is validated according to the rules of “time of day” validation, except the hour component (H) can be a number between 1 and 24 inclusive to support 24-hour time values.

- timestamp

  A field defined as a timestamp field is validated according to the rules of “time” validation for the time component and according to the rules of “date” validation for the date component.

The FileMaker class cannot pre-validate all of the field validation options that are available in FileMaker Pro. The following validation options cannot be pre-validated because they are dependent on the state of all the data in the database at the time that the data is committed:

- unique value
- existing value
- in range
- member of value list
- validate by calculation

Pre-validating records in a command

For a command object, use the `validate()` method to validate one field or the entire command against the pre-validation rules enforceable by the PHP engine. If you pass the optional field name argument, only that field is pre-validated.

If the pre-validation passes, then the `validate()` method returns TRUE. If the pre-validation fails, then the `validate()` method returns a FileMaker_Error_Validation object containing details about what failed to validate.

Pre-validating records

For a record object, use the `validate()` method to validate one field or all the fields in the record against the pre-validation rules enforceable by the PHP engine. If you pass the optional field name argument, only that field is pre-validated.

If the pre-validation passes, then the `validate()` method returns TRUE. If the pre-validation fails, then the `validate()` method returns a FileMaker_Error_Validation object containing details about what failed to validate.

Pre-validating fields

For a field object, use the `validate()` method to determine whether a given value is valid for a field.

If the pre-validation passes, then the `validate()` method returns TRUE. If the pre-validation fails, then the `validate()` method returns a FileMaker_Error_Validation object containing details about what failed to validate.

Processing the validation errors

When pre-validation fails, the FileMaker_Error_Validation object returned contains a three-element array for each validation failure:

1. The field object that failed pre-validation

2. A validation constant value that indicates the validation rule that failed:

1 - FILEMAKER_RULE_NOTEMPTY

2 - FILEMAKER_RULE_NUMERICONLY

3 - FILEMAKER_RULE_MAXCHARACTERS

4 - FILEMAKER_RULE_FOURDIGITYEAR

5 - FILEMAKER_RULE_TIMEOFDAY

6 - FILEMAKER_RULE_TIMESTAMP_FIELD

7 - FILEMAKER_RULE_DATE_FIELD

8 - FILEMAKER_RULE_TIME_FIELD

3. The actual value entered for the field that failed pre-validation

You can also use the following methods with a FileMaker_Error_Validation object:

- Use the `isValidationError()` method to test whether the error is a validation error.
- Use the `numErrors()` method to get the number of validation rules that failed.

Example

```
//Create an Add request
$addrequest = $fm->newAddCommand('test', array('join' => 'added', 'maxchars' =>
'abcx', 'field' => 'something' , 'numericonly' => 'abc'));

//Validate all fields
$result = $addrequest->validate();

//If the validate() method returned any errors, print the name of the field, the
error number, and the value that failed.
if(FileMaker::isError($result)){
    echo 'Validation failed:'. "\n";
    $validationErrors= $result->getErrors();
    foreach ($validationErrors as $error) {
        $field =  $error[0];
        echo 'Field Name: ' .  $field->getName(). "\n";
        echo 'Error Code: ' .  $error[1] . "\n";
        echo 'Value: ' .  $error[2] . "\n";
    }
}
```

Output

```
Validation failed:
Field Name: numericonly
Error Code: 2
Value: abc
Field Name: maxchars
Error Code: 3
Value: abcx
```

Handling errors

The FileMaker class defines the FileMaker_Error object to help you handle errors that occur in a PHP solution.

An error can occur when a command runs. If an error does occur, the command returns a FileMaker_Error object. It is a good practice to check the error that is returned when a command runs.

Use the following methods to learn more about the error indicated in the FileMaker_Error object.

- Test for whether a variable is a FileMaker Error object by calling the `isError()` method.
- Get the number of errors that occurred by calling the `numErrors()` method.
- Retrieve an array of arrays describing the errors that occurred by calling the `getErrors()` method.
- Display an error message by calling the `getMessage()` method.

Example

```
$result = $findCommand->execute();
if (FileMaker::isError($result)) {
        echo "<p>Error: " . $result->getMessage() . "</p>";
        exit;
}
```

For information about the error codes returned with the FileMaker Error object, see appendix A, “Error codes for Custom Web Publishing with PHP.”

Chapter 6
Staging, testing, and monitoring a site

This chapter provides instructions for staging and testing a Custom Web Publishing site before deploying it in a production environment. Instructions are also provided for using log files to monitor the site during testing or after deployment.

Staging a Custom Web Publishing site

Before you can properly test your site, copy or move the required files to the correct locations on the staging server(s).

To stage your site and prepare it for testing:

1. Complete all of the steps in chapter 3, “Preparing databases for Custom Web Publishing.”
2. Check that Custom Web Publishing with PHP has been enabled and properly configured in the FileMaker Server Admin Console.

   Note For instructions, see FileMaker Server Help.

3. Verify that the web server and the Web Publishing Engine are running.
4. Copy or move your site files to the web server component of your FileMaker Server deployment.

   Copy or move your site files to the following directory on the web server machine:

   - IIS (Windows): `<drive>:\Inetpub\wwwroot` where `<drive>` is the drive on which the Web Publishing Engine component of your FileMaker server deployment resides.
   - Apache (Mac OS): `/Library/WebServer/Documents`

5. If you have not already done so, copy or move any referenced container objects to the appropriate directory on the web server machine.
   - If the database file is properly hosted and accessible on the Database Server component of the FileMaker Server deployment, and the container fields store the actual files in the FileMaker database, then you don’t need to relocate the container field contents.
   - If a database container field stores a file reference instead of an actual file, then the referenced container object must be stored in the FileMaker Pro Web folder when the record is created or edited. To stage your site, you must copy or move the referenced containers to a folder with the same relative location in the root folder of the web server software.
   - If a database container field stores the container object externally, then use the Upload Database assistant to transfer the database file and container field objects from your computer’s file system to FileMaker Server. If you manually upload a database that uses a container field with externally stored objects, then you must copy or move the referenced objects into a subfolder of the RC_Data_FMS folder, as described in “Container fields with externally stored data” on page 16.

   See “Publishing the contents of container fields on the web” on page 15.

6. Begin testing your site.

Testing a Custom Web Publishing site

Before notifying users that your Custom Web Publishing site is available, verify that it displays and functions as expected.

- Test features like finding, adding, deleting, and sorting records with different accounts and privilege sets.
- Verify that privilege sets are performing as expected by logging in with different accounts. Make sure unauthorized users can't access or modify your data.
- Check all scripts to verify that the outcome is expected. See "FileMaker scripts and Custom Web Publishing" on page 18 for information on designing web-friendly scripts.
- Test your site with different operating systems and web browsers.
- When creating solutions that use the FileMaker API for PHP, it is recommended that you build your solutions with cookie support enabled. The FileMaker API for PHP has better response times with cookies enabled. Cookies are not required to use Custom Web Publishing features, but cookies do allow the Web Publishing Engine to cache session information.

Note If you have installed the web server, Web Publishing Engine, and the Database Server in a single-machine deployment, you can view and test your site without using a network connection. Move your site files to the appropriate directory on that machine, and enter the following URL in your browser:

```
http://127.0.0.1/<site_path>
```

where `<site_path>` is the relative path to the homepage for your site.

Monitoring your site

Use the following types of log files to monitor your Custom Web Publishing site and gather information about web users who visit your site:

- Web server access and error logs
- Web Publishing Engine log
- Web Server Module error log
- Tomcat logs

Using the web server access and error logs

- IIS (Windows): The Microsoft IIS web server generates an access log file and displays errors in the Windows Event Viewer instead of writing them to a log file. The access log file, which is in the W3C Extended Log File Format by default, is a record of all incoming HTTP requests to the web server. You can also use the W3C Common Logfile Format for the access log. For more information, see the documentation for the Microsoft IIS web server.
- Apache (Mac OS only): The Apache web server generates an access log file and an error log file. The Apache access log file, which is in the W3C Common Logfile Format by default, is a record of all incoming HTTP requests to the web server. The Apache error log is a record of problems involving processing HTTP requests. For more information on these log files, see the documentation for the Apache web server.

Note For information on the W3C Common Logfile Format and the W3C Extended Log File Format, see the World Wide Web Consortium website at http://www.w3.org.

Using the Web Publishing Engine log

By default, the Web Publishing Engine generates a log file called wpe.log that contains a record of any Web Publishing Engine errors that have occurred, including application errors, usage errors, and system errors. You can also have the Web Publishing Engine include information related to Custom Web Publishing, such as end-user XML requests to generate web publishing output or changes to the Custom Web Publishing settings.

Because the FileMaker API for PHP uses HTTP POST to access the Web Publishing Engine, the wpe.log file does not log details about the PHP requests. You can use the wpe.log file to see when users made PHP requests by looking at the XML requests that are logged.

The wpe.log file is located on the Web Publishing Engine component of the FileMaker Server deployment:

- IIS (Windows):
  `<drive>:\Program Files\FileMaker\FileMaker Server\Logs\wpe.log`
  where `<drive>` is the primary drive from which the system is started.
- Apache (Mac OS): `/Library/FileMaker Server/Logs/wpe.log`

Web Publishing Engine log settings

The wpe.log file is generated if the **Enable logging for Custom Web Publishing** option is enabled in the Admin Console.

Logging option enabled	Information recorded in wpe.log
Error level messages	Any Web Publishing Engine errors that have occurred, including application errors, usage errors, and system errors.
Info and Error Level messages	Any errors as described above, and information about access to the Web Publishing Engine. It contains a record of all end-user XML requests to generate Custom Web Publishing output.

The **Error level messages** setting is enabled by default. For information on setting these options using the Admin Console, see FileMaker Server Help.

Note For Custom Web Publishing with FileMaker Server 12, the wpe.log file replaces the wpc_access_log.txt and pe_application_log.txt files used in previous releases.

Important Over time, the wpe.log file may become very large. Use the Admin Console to set the maximum size for the wpe.log file. When the wpe.log file reaches this maximum size, the Web Publishing Engine copies the wpe.log file to a single backup file, wpe.log.1, and creates a new wpe.log file. You may wish to save an archive of the wpe.log.1 file on a regular basis, if you want more than one backup copy.

Web Publishing Engine log format

The wpe.log file uses the following format for each entry:

```
[TIMESTAMP_GMT] [WPC_HOSTNAME] [CLIENT_IP:PORT] [ACCOUNT_NAME] [MODULE_TYPE]
[SEVERITY] [FM_ERRORCODE] [RETURN_BYTES] [MESSAGE]
```

where:

- [TIMESTAMP_GMT] is the date and time of the entry, in Greenwich Mean Time (GMT).
- [WPC_HOSTNAME] is the machine name for the machine where the Web Publishing Engine is installed.
- [CLIENT_IP:PORT] is the IP address and port of the client where the XML request originated.
- [ACCOUNT_NAME] is the account name used for logging into the hosted FileMaker database.
- [MODULE_TYPE] is either: XML, for Custom Web Publishing with XML requests, or PHP, for Custom Web Publishing with PHP requests.
- [SEVERITY] is either INFO, indicating an informational message, or ERROR, indicating an error message.
- [FM_ERROR_CODE] is the error number returned for an error message. The error number may be an error code for FileMaker databases (see “Error code numbers for FileMaker databases” on page 48).

  In addition, the error number may be an HTTP error number, prefixed by an “HTTP:” string.
- [RETURN_BYTES] is the number of bytes returned by the request.
- [MESSAGE] provides additional information about the log entry.

Web Publishing Engine log message examples

The following examples show the types of messages that may be included in the wpe.log file:

- When the Web Publishing Engine starts and stops

```
2012-06-02 15:15:31 -0700 - - - - INFO - - FileMaker Server
Web Publishing Engine started.
2012-06-02 15:46:52 -0700 - - - - INFO - - FileMaker Server
Web Publishing Engine stopped.
```

- Successful or failed XML query requests

```
2012-06-02 15:21:08 -0700 WPC_SERVER 192.168.100.101:0 jdoe XML
INFO 0 3964 "/fmi/xml/fmresultset.xml?-db=Contacts&-
lay=Contact_Details&-findall"
2012-06-02 15:26:31 -0700 WPC_SERVER 192.168.100.101:0 jdoe XML
ERROR 5 596 "/fmi/xml/fmresultset.xml?-db=Contacts&-
layout=Contact_Details&-findall"
```

- Scripting errors

```
2012-06-02 17:33:12 -0700 WPC_SERVER 192.168.100.101:0 jdoe - ERROR
4 - Web Scripting Error: 4, File: "10b_MeetingsUpload", Script: "OnOpen",
Script Step: "Show Custom Dialog"
```

- Changes to the Custom Web Publishing settings

```
2012-06-09 10:59:49 -0700 WPC_SERVER 192.168.100.101:0 jdoe - INFO
- - XML Web Publishing Engine is enabled.
```

- System errors

```
2012-06-02 15:30:42 -0700 WPC_SERVER 192.168.100.101:0 jdoe XML
ERROR - - Communication failed
```

Using the Web Server Module error log

If the web server is unable to connect to the Web Publishing Engine, the Web Server Module generates a log file that records any errors with its operation. This log file is named web_server_module_log.txt, and is located on the web server component of the FileMaker Server deployment:

- IIS (Windows): `<drive>:\Program Files\FileMaker\FileMaker Server\Logs\web_server_module_log.txt`
  where `<drive>` is the primary drive from which the system is started.
- Apache (Mac OS): `/Library/FileMaker Server/Logs/web_server_module_log.txt`

Using the Tomcat logs

When FileMaker Server has a problem caused by an internal web server error, you may find it helpful to view the Tomcat logs. The Tomcat logs are located on the web server component of the FileMaker Server deployment:

- IIS (Windows): `<drive>:\Program Files\FileMaker\FileMaker Server\Admin\admin-master-tomcat\logs/`
  where `<drive>` is the primary drive from which the system is started.
- Apache (Mac OS): `/Library/FileMaker Server/Admin/admin-master-tomcat/logs/`

Troubleshooting your site

If you have trouble viewing or using your site, verify the following:

- The extended privileges in the database are set for Custom Web Publishing with PHP and assigned to a user account. See “Enabling Custom Web Publishing with PHP for databases” on page 13.
- The database is hosted and opened by FileMaker Server. See FileMaker Server Help.
- The database account name and password you are using, if any, are correct.
- The web server and the Web Publishing Engine are running.
- PHP Publishing is enabled in the Web Publishing Engine.
  - Open the FileMaker Server Technology Tests page in a browser:

    `http://<server>:16000/test`

    where `<server>` is the machine on which the FileMaker Server resides.
  - Click the link **Test PHP Custom Web Publishing** to open a PHP page that accesses the FMServer_Sample test database.

For more information, see the *FileMaker Server Getting Started Guide* and FileMaker Server Help.

Appendix A
Error codes for Custom Web Publishing with PHP

The Web Publishing Engine supports two types of error codes that can occur for Custom Web Publishing:

- Database and data request errors. The Web Publishing Engine generates an error code whenever data is requested from published database. The FileMaker API for PHP returns this error code as a FileMaker_Error object. See the next section, “Error code numbers for FileMaker databases.”
- PHP errors. These errors are generated and returned by PHP components, including the cURL module. See “Error code numbers for PHP components” on page 55.

Error code numbers for FileMaker databases

It is up to you, as the developer of the Custom Web Publishing solution, to check the value of the returned error code and handle it appropriately. The Web Publishing Engine does not handle database errors.

Error Number	Description
-1	Unknown error
0	No error
1	User canceled action
2	Memory error
3	Command is unavailable (for example, wrong operating system, wrong mode, etc.)
4	Command is unknown
5	Command is invalid (for example, a Set Field script step does not have a calculation specified)
6	File is read-only
7	Running out of memory
8	Empty result
9	Insufficient privileges
10	Requested data is missing
11	Name is not valid
12	Name already exists
13	File or object is in use
14	Out of range
15	Can’t divide by zero
16	Operation failed, request retry (for example, a user query)
17	Attempt to convert foreign character set to UTF-16 failed
18	Client must provide account information to proceed

Error Number	Description
19	String contains characters other than A-Z, a-z, 0-9 (ASCII)
20	Command or operation cancelled by triggered script
100	File is missing
101	Record is missing
102	Field is missing
103	Relationship is missing
104	Script is missing
105	Layout is missing
106	Table is missing
107	Index is missing
108	Value list is missing
109	Privilege set is missing
110	Related tables are missing
111	Field repetition is invalid
112	Window is missing
113	Function is missing
114	File reference is missing
115	Menu set is missing
116	Layout object is missing
117	Data source is missing
118	Theme is missing
130	Files are damaged or missing and must be reinstalled
131	Language pack files are missing (such as template files)
200	Record access is denied
201	Field cannot be modified
202	Field access is denied
203	No records in file to print, or password doesn’t allow print access
204	No access to field(s) in sort order
205	User does not have access privileges to create new records; import will overwrite existing data
206	User does not have password change privileges, or file is not modifiable
207	User does not have sufficient privileges to change database schema, or file is not modifiable
208	Password does not contain enough characters
209	New password must be different from existing one
210	User account is inactive
211	Password has expired
212	Invalid user account and/or password. Please try again
213	User account and/or password does not exist
214	Too many login attempts

Error Number	Description
215	Administrator privileges cannot be duplicated
216	Guest account cannot be duplicated
217	User does not have sufficient privileges to modify administrator account
218	Password and verify password do not match
300	File is locked or in use
301	Record is in use by another user
302	Table is in use by another user
303	Database schema is in use by another user
304	Layout is in use by another user
306	Record modification ID does not match
307	Transaction could not be locked because of a communication error with the host
308	Theme is in use by another user
400	Find criteria are empty
401	No records match the request
402	Selected field is not a match field for a lookup
403	Exceeding maximum record limit for trial version of FileMaker Pro
404	Sort order is invalid
405	Number of records specified exceeds number of records that can be omitted
406	Replace/Reserialize criteria are invalid
407	One or both match fields are missing (invalid relationship)
408	Specified field has inappropriate data type for this operation
409	Import order is invalid
410	Export order is invalid
412	Wrong version of FileMaker Pro used to recover file
413	Specified field has inappropriate field type
414	Layout cannot display the result
415	One or more required related records are not available
416	A primary key is required from the data source table
417	Database is not a supported data source
500	Date value does not meet validation entry options
501	Time value does not meet validation entry options
502	Number value does not meet validation entry options
503	Value in field is not within the range specified in validation entry options
504	Value in field is not unique as required in validation entry options
505	Value in field is not an existing value in the database file as required in validation entry options
506	Value in field is not listed on the value list specified in validation entry option
507	Value in field failed calculation test of validation entry option
508	Invalid value entered in Find mode

Error Number	Description
509	Field requires a valid value
510	Related value is empty or unavailable
511	Value in field exceeds maximum number of allowed characters
512	Record was already modified by another user
600	Print error has occurred
601	Combined header and footer exceed one page
602	Body doesn't fit on a page for current column setup
603	Print connection lost
700	File is of the wrong file type for import
706	EPSF file has no preview image
707	Graphic translator cannot be found
708	Can't import the file or need color monitor support to import file
709	QuickTime movie import failed
710	Unable to update QuickTime file reference because the database file is read-only
711	Import translator cannot be found
714	Password privileges do not allow the operation
715	Specified Excel worksheet or named range is missing
716	A SQL query using DELETE, INSERT, or UPDATE is not allowed for ODBC import
717	There is not enough XML/XSL information to proceed with the import or export
718	Error in parsing XML file (from Xerces)
719	Error in transforming XML using XSL (from Xalan)
720	Error when exporting; intended format does not support repeating fields
721	Unknown error occurred in the parser or the transformer
722	Cannot import data into a file that has no fields
723	You do not have permission to add records to or modify records in the target table
724	You do not have permission to add records to the target table
725	You do not have permission to modify records in the target table
726	There are more records in the import file than in the target table. Not all records were imported
727	There are more records in the target table than in the import file. Not all records were updated
729	Errors occurred during import. Records could not be imported
730	Unsupported Excel version (convert file to Excel 2000 format or a later supported version and try again)
731	File you are importing from contains no data
732	This file cannot be inserted because it contains other files
733	A table cannot be imported into itself
734	This file type cannot be displayed as a picture
735	This file type cannot be displayed as a picture. It will be inserted and displayed as a file
736	There is too much data to be exported to this format. It will be truncated.
737	Bento table you are importing is missing

Error Number	Description
800	Unable to create file on disk
801	Unable to create temporary file on System disk
802	Unable to open file. This error can be cause by one or more of the following: ▪ Invalid database name ▪ File is closed in FileMaker Server ▪ Invalid permission
803	File is single user or host cannot be found
804	File cannot be opened as read-only in its current state
805	File is damaged; use Recover command
806	File cannot be opened with this version of FileMaker Pro
807	File is not a FileMaker Pro file or is severely damaged
808	Cannot open file because access privileges are damaged
809	Disk/volume is full
810	Disk/volume is locked
811	Temporary file cannot be opened as FileMaker Pro file
813	Record Synchronization error on network
814	File(s) cannot be opened because maximum number is open
815	Couldn’t open lookup file
816	Unable to convert file
817	Unable to open file because it does not belong to this solution
819	Cannot save a local copy of a remote file
820	File is in the process of being closed
821	Host forced a disconnect
822	FMI files not found; reinstall missing files
823	Cannot set file to single-user, guests are connected
824	File is damaged or not a FileMaker file
825	File is not authorized to reference the protected file
826	File path specified is not a valid file path
850	Path is not valid for the operating system
851	Cannot delete an external file from disk
852	Cannot write a file to the external storage
900	General spelling engine error
901	Main spelling dictionary not installed
902	Could not launch the Help system
903	Command cannot be used in a shared file
904	Command can only be used in a file hosted under FileMaker Server
905	No active field selected; command can only be used if there is an active field
906	Current file is not shared; command can be used only if the file is shared

Error Number	Description
920	Can’t initialize the spelling engine
921	User dictionary cannot be loaded for editing
922	User dictionary cannot be found
923	User dictionary is read-only
951	An unexpected error occurred
954	Unsupported XML grammar
955	No database name
956	Maximum number of database sessions exceeded
957	Conflicting commands
958	Parameter missing in query
959	Custom Web Publishing technology is disabled
960	Parameter is invalid
1200	Generic calculation error
1201	Too few parameters in the function
1202	Too many parameters in the function
1203	Unexpected end of calculation
1204	Number, text constant, field name or "(" expected
1205	Comment is not terminated with "*/"
1206	Text constant must end with a quotation mark
1207	Unbalanced parenthesis
1208	Operator missing, function not found or "(" not expected
1209	Name (such as field name or layout name) is missing
1210	Plug-in function has already been registered
1211	List usage is not allowed in this function
1212	An operator (for example, +, -, *) is expected here
1213	This variable has already been defined in the Let function
1214	AVERAGE, COUNT, EXTEND, GETREPETITION, MAX, MIN, NPV, STDEV, SUM and GETSUMMARY: expression found where a field alone is needed
1215	This parameter is an invalid Get function parameter
1216	Only Summary fields allowed as first argument in GETSUMMARY
1217	Break field is invalid
1218	Cannot evaluate the number
1219	A field cannot be used in its own formula
1220	Field type must be normal or calculated
1221	Data type must be number, date, time, or timestamp
1222	Calculation cannot be stored
1223	Function referred to is not yet implemented
1224	Function referred to does not exist
1225	Function referred to is not supported in this context

Error Number	Description
1300	The specified name can't be used
1400	ODBC client driver initialization failed; make sure the ODBC client drivers are properly installed
1401	Failed to allocate environment (ODBC)
1402	Failed to free environment (ODBC)
1403	Failed to disconnect (ODBC)
1404	Failed to allocate connection (ODBC)
1405	Failed to free connection (ODBC)
1406	Failed check for SQL API (ODBC)
1407	Failed to allocate statement (ODBC)
1408	Extended error (ODBC)
1409	Extended error (ODBC)
1410	Extended error (ODBC)
1411	Extended error (ODBC)
1412	Extended error (ODBC)
1413	Extended error (ODBC)
1414	SQL statement is too long
1450	Action requires PHP privilege extension
1451	Action requires that current file be remote
1501	SMTP authentication failed
1502	Connection refused by SMTP server
1503	Error with SSL
1504	SMTP server requires the connection to be encrypted
1505	Specified authentication is not supported by SMTP server
1506	Email message(s) could not be sent successfully
1507	Unable to log in to the SMTP server
1550	Cannot load the plug-in or the plug-in is not a valid plug-in
1551	Cannot install the plug-in. Cannot delete an existing plug-in or cannot write to the folder or disk
1626	Protocol is not supported
1627	Authentication failed
1628	There was an error with SSL
1629	Connection timed out; the timeout value is 60 seconds
1630	URL format is incorrect
1631	Connection failed

Error code numbers for PHP components

The FileMaker API for PHP makes use of several PHP components. These PHP components may return additional error codes that are not listed above.

For example, if the Web Publishing Core or FileMaker Server services are not running, you may receive the cURL module error CURLE_GOT_NOTHING (52).

For information on PHP related error codes, see the PHP website at http://php.net.

Index

G

H

I

J

L

M

N

O

P

Q

R

S

T

U

V

W

X