FileMaker® Server 16

Custom Web Publishing Guide

FileMaker®
An Apple Subsidiary

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

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 using FileMaker® Pro to create databases. You should understand the basics of FileMaker Pro database design and the concepts of fields, relationships, layouts, portals, and containers. For information about FileMaker Pro, see FileMaker Pro Help.

This guide also assumes you are experienced with developing websites, especially with using technologies like XML or PHP to integrate FileMaker data with websites and web applications.

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

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

Where to find FileMaker documentation

- In FileMaker Server Admin Console, choose **Help** menu > **FileMaker Server Product Documentation**.
- Click the links on the FileMaker Server Admin Console Start Page.
- To learn about, view, or download additional FileMaker documentation, visit http://www.filemaker.com/documentation.

Online Help is accessible from FileMaker Server Admin Console. Choose **Help** menu > **FileMaker Server Help**.

Chapter 1
Introducing Custom Web Publishing

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

FileMaker WebDirect: With FileMaker WebDirect, you can quickly and easily publish layouts from a database on the web. You don't need to install additional software—with compatible web browser software and access to the Internet or an intranet, web users can connect to your FileMaker WebDirect solution to view, edit, sort, or search records, if you give them access privileges.

With FileMaker WebDirect, the host computer must be running FileMaker Server. The user interface resembles the desktop FileMaker Pro application. The webpages and forms that the web user interacts with are dependent on the layouts and views defined in the FileMaker Pro database. See FileMaker WebDirect 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 webpage that you can further customize with HTML. The webpage doesn't change when information in your database changes, and users don't connect to your database. (With FileMaker WebDirect, the data is updated in the web browser whenever the data is updated in the database.) See FileMaker Pro Help.

FileMaker Data API: If you're experienced with using Representational State Transfer (REST) architecture, FileMaker provides a REST API implementation that allows web services to access data in hosted solutions. Your web service calls the FileMaker Data API to obtain an authentication token for access to a hosted solution, then uses that token in subsequent calls to create records, update records, delete records, and perform find requests. The FileMaker Data API returns data in JavaScript Object Notation (JSON). See FileMaker Data API Guide.

Custom Web Publishing: To integrate your FileMaker database with a custom website, 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:

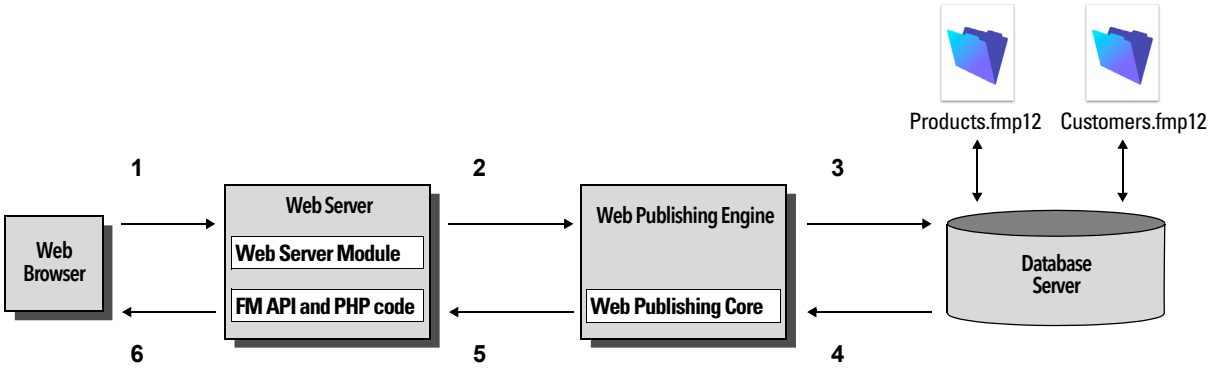
- Custom Web Publishing with XML: Use XML data publishing to exchange FileMaker data with other websites and applications. By using HTTP URL requests with FileMaker query commands and parameters, you can query a database hosted by FileMaker Server, download the resulting data in XML format, and use the resulting XML data in whatever way you want.
- 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 webpages yourself, you have complete control over the user interface and how the user interacts with the data.

About the Web Publishing Engine

To support FileMaker WebDirect 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 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 FileMaker API for PHP converts the PHP request into an XML request. The Web Publishing Engine processes the XML request and sends XML data back to the FileMaker API for PHP. The FileMaker API for PHP then converts the XML data into PHP objects that can be used by the PHP application.
   - 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 Security Guide.

For information about getting a database ready for Custom Web Publishing, see chapter 2, “Preparing databases for Custom Web Publishing.”

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 to 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.

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.

Comparing XML to PHP

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

Reasons to choose XML

- FileMaker XML request parameter syntax is designed for database interaction, which simplifies 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 exporting XML from FileMaker Pro databases.

Note For information about Custom Web Publishing with XML, see chapter 3, “About Custom Web Publishing with XML.”

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 php.net.
- PHP enables access to a wide variety of third-party components that you can integrate into your solutions.

Note For information about Custom Web Publishing with PHP, see chapter 6, “About Custom Web Publishing with PHP.”

Chapter 2
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 in a database

You must enable a Custom Web Publishing extended privilege in each database you want to publish. If you don't enable a Custom Web Publishing extended privilege in the database, web users won't be able to use Custom Web Publishing to access the database even if it is hosted by 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 that has the Full Access privilege set. Alternatively, you can open the database using an account that has the Manage Extended Privileges access privileges.
2. Assign the Custom Web Publishing extended privilege that you want to use:
   - For Custom Web Publishing with XML, use fmxml
   - For Custom Web Publishing with PHP, use fmphp
3. Assign the privilege set(s) that include the Custom Web Publishing extended privilege to one or more accounts, or to the Admin or Guest account.

Note 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 punctuation characters such as "**!**" and "**%**," but do not include colons. For information on setting up accounts, see FileMaker Pro Help.

Accessing a protected database

Custom Web Publishing enables you to restrict access to your published databases through database password protection, database encryption, and secure connections. When using a Custom Web Publishing solution to access a database, web users may be prompted for their account information. If the Guest account for the database is disabled or does not have a privilege set enabled that includes a Custom Web Publishing extended privilege, the Web Publishing Engine uses HTTP Basic Authentication to request authentication from web users. The web user's browser displays the HTTP Basic Authentication dialog box for the user to enter a user name and password for an account that has a Custom Web Publishing extended privilege.

The following list summarizes the process that occurs when a web user uses a Custom Web Publishing solution to access a database:

- If you have not assigned a password for an account, web users only specify the account name.
- If the Guest account is disabled, then users will be prompted for account name and password when they access the database. The account must have a Custom Web Publishing extended privilege enabled.
- If the Guest account is enabled and has a privilege set enabled that includes a Custom Web Publishing extended privilege, all web users automatically open the database with the access privileges assigned to the Guest account. If the Custom Web Publishing extended privilege is assigned to the Guest account:
  - Web users are not prompted for an account name and password when opening a file.
  - All web users will automatically log in with the Guest account and assume the Guest account privileges. You can let users change their login accounts from a web browser with the Re-Login script step (for example, to switch from the Guest account to an account with more 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.
- When a web user has entered valid account information, that account information is reused as long as the browser session has not timed out. When the browser session times out, then the web user is again prompted to enter a valid account.

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

Protecting your published databases

When using Custom Web Publishing, you can limit who can access your published databases.

- Assign passwords to database accounts that are used for Custom Web Publishing.
- Enable a Custom Web Publishing extended privilege only in the privilege sets for accounts that you want to allow access to your published databases.
- Disable the Custom Web Publishing extended privilege for a specific database by deselecting the fmxml or 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 Installation and Configuration 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, you can 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.

FileMaker Server supports encryption for data written to disk and for data transmitted to clients.

- Encrypt your database by using the Database Encryption feature of FileMaker Pro Advanced. Encryption protects the FileMaker database file and any temporary files written to disk. See FileMaker Server Installation and Configuration Guide and FileMaker Pro Help.
  - An encrypted database that is hosted on FileMaker Server is opened by using Admin Console or the command line interface (CLI). As the FileMaker Server administrator, you open the file with its database encryption password, so that FileMaker clients can use the encrypted database.
  - Once the FileMaker encrypted database is opened with the encryption password by the FileMaker Server administrator, FileMaker clients don't need the encryption password to access the encrypted database. For information about opening an encrypted database, see FileMaker Server Help.
- Use Secure Sockets Layer (SSL) encryption for communication between the web server and web browsers. SSL connections are accessed through an HTTPS connection. FileMaker Server provides a standard SSL certificate signed by FileMaker, Inc. that does not verify the server name. The FileMaker default certificate is intended only for test purposes. A custom SSL certificate is required for production use. See FileMaker Server Installation and Configuration Guide.

  If you enable **Use HSTS for web clients** in Admin Console, use the HTTPS directory for hosting PHP site files. See chapter 7, "General steps for Custom Web Publishing with PHP."

For information on securing your database, see FileMaker Security Guide.

Note For security reasons, webpages hosted by other web servers may not use the <iframe> tag to embed Custom Web Publishing content. If you want to embed Custom Web Publishing content in the <iframe> tags of separate webpages, those webpages must be hosted by the FileMaker Server web server.

Web server support for Internet media types (MIME)

Your web server determines the support for the current MIME (Multipurpose Internet Mail Extensions) types registered for the Internet. The Web Publishing Engine does not change a web server's support for MIME. See the documentation for your web server.

About 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 field objects embedded in a database

If a container field stores the actual files in the FileMaker database, then you don't need to do anything with the container field contents if the database file is properly hosted and accessible on FileMaker Server. See "About the URL syntax for FileMaker container objects in XML solutions" on page 26.

Container fields with stored file references

If a container field stores a file reference, then you must 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]:\Program Files\FileMaker\FileMaker Server\HTTPServer\conf` where `[drive]` is the drive on which the Web Publishing Engine component of your FileMaker Server deployment resides.
   - For Apache (macOS): `/Library/FileMaker Server/HTTPServer/htdocs`

Note For container objects stored as file references, your web server must be configured to support the MIME 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. See the documentation for your web server.

Container fields with externally stored data

If a container field stores objects externally—that is, if you selected **Store container data externally** in the FileMaker Pro Options for Field dialog box—use FileMaker Pro to transfer database files from the client file system to FileMaker Server. When you use FileMaker Pro to upload a database, the externally stored container field data is uploaded to FileMaker Server as part of the process. See FileMaker Pro Help for information on transferring the database files to FileMaker Server.

When 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 the 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 (macOS) to those files—in the following folders:
   - Windows: `[drive]:\Program Files\FileMaker\FileMaker Server\Data\Databases\` where `[drive]` is the primary drive from which the system is started.
   - macOS: `/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 macOS, change the files to belong to the **fmsadmin** group.

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

Container fields and progressive download

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 option to optimize for web viewing.

When the FileMaker Server setting **Use SSL for database connections** is selected, FileMaker Server uses secure connections to transmit data over HTTPS.

- Interactive container data is downloaded over HTTPS.
- The data is as secure as if the hosted solution were a local database, since no temporary cache files are created and the data is encrypted during transmission.

When the FileMaker Server setting **Use SSL for database connections** is not selected, the connections that FileMaker Server uses to transmit data are not encrypted during transmission and data is transmitted over HTTP.

- FileMaker clients see the interactive container data with minimal delay.
- FileMaker Server caches the container field data to a cache folder on the server when a FileMaker Pro, FileMaker Go, or web client requests the data. The data may remain in the cache folder on the server for two hours, until FileMaker Server periodically empties the cache folder. The data is not cached locally on the client.

Restart the FileMaker Server service (Windows) or FileMaker Server background processes (macOS) when the **Use SSL for database connections** setting is changed in order for the new settings to take effect.

How web users view container field data

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 and combine several tasks. When used with Custom Web Publishing, FileMaker scripts allow web users to perform more tasks or a series of tasks.

FileMaker supports many script steps in Custom Web Publishing. Web users can perform a variety of automated tasks when you use scripts in a query string for a URL. To see script steps that Custom Web Publishing supports, in the FileMaker Pro Script Workspace window, click the **Compatibility** button and choose **Custom Web Publishing**. Script steps that are not dimmed are 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 19. 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. Check the Web Publishing Engine log file (wpe.log) for any scripting-related errors. See “Using the Web Publishing Engine log” on page 94.

Keep these tips and considerations in mind:

- Consider what values a script may return. Be prepared to handle all of the data that is returned. In FileMaker Pro, a script may return all the records from a table or from the current found set. But if a script returns all the records from a table, a web application may run out of memory trying to processing the records. Consider using the `-max` query parameter with XML queries or the `setRange()` method with PHP queries to limit the number of records returned.
- 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 Script Workspace window, grant full access privileges to a script to allow the script to perform tasks that you would not grant individuals access to. For example, you can prevent users from deleting records with their accounts and privileges, but still allow them to run a script that would delete certain types of records under conditions predefined within a script.
- To allow scripts to install plug-ins for Custom Web Publishing and FileMaker WebDirect solutions, use FileMaker Server Admin Console to enable the setting **Allow Install Plug-In File script step to update plug-ins for web publishing**. To prevent script from installing plug-ins for web publishing solutions, clear this setting.
- Some scripts that work with one step from a FileMaker Pro client may require an additional Commit Record/Request script 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 script step, because data changes aren't visible in the browser until the data is saved or "submitted" to the server. This includes several script steps like Cut, Copy, and Paste. 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 a "Web Publishing Engine" string, then you know that the current user is accessing your database with Custom Web Publishing. For information on functions, see FileMaker Pro Help.
- Open each script that web users might run, and verify that the script will execute properly when the database is hosted as a Custom Web Publishing solution. Check that the script uses only script steps that are supported for Custom Web Publishing, as described above.

Script behavior in Custom Web Publishing solutions

Some script steps function differently in Custom Web Publishing solutions than in FileMaker Pro. See FileMaker Pro Help for compatibility information.

Scripts in Custom Web Publishing solutions cannot perform scripts in other FileMaker files unless the files are hosted on the same installation of FileMaker Server and unless the same Custom Web Publishing extended privilege is enabled in the other files.

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 information on script triggers, see FileMaker Pro Help.

Note In Custom Web Publishing solutions, the OnFirstWindowOpen and OnLastWindowClose script triggers are not activated. Run scripts manually using the XML –script query parameter or the PHP `newPerformScriptCommand()` method.

Chapter 3
About Custom Web Publishing with XML

Creating dynamic websites with the Web Publishing Engine

The Web Publishing Engine provides Custom Web Publishing for FileMaker Server using XML data publishing. Custom Web Publishing provides several benefits:

- **Customization**: You can determine how web users interact with FileMaker data, and how the data displays in web browsers.
- **Data interchange**: By using FileMaker XML, you can exchange FileMaker data with other websites and applications.
- **Data integration**: You can integrate FileMaker data into other websites, with other middleware, and with custom applications. You can make the data look like it belongs to another website instead of displaying an entire FileMaker layout in the web browser.
- **Security**: The FileMaker Server administrator can individually enable or disable XML web publishing for all databases hosted by the server. As the FileMaker database owner, you can control web user access to or XML web publishing for each database.
- **Control and filtering of published data**: You can control and filter the data and the type of database information you want to publish, which prevents unauthorized use of the database. You can also hide metadata, such as database and field names.
- **Based on an open standard**: You have more access to tools, resources and skilled personnel for Custom Web Publishing solutions. If you know standard XML, then you can start developing solutions after learning a few unique details about Custom Web Publishing with XML, such as the URL syntax and query parameters to use.

Custom Web Publishing with XML allows you to retrieve data from FileMaker databases, and easily use the data in other output formats. By using an HTTP request with the appropriate query commands and parameters, you can retrieve FileMaker data as an XML document. You can then use the XML data in other applications. See “Accessing XML data via the Web Publishing Engine” on page 28.

Key features in Custom Web Publishing with XML

FileMaker Server Custom Web Publishing with XML provides several important features:

- Databases are hosted on FileMaker Server, and FileMaker Pro is not required to be running.
- You can use server-side processing of the XML using JavaScript.
- 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.
- Web users can perform complex, multistep scripts. FileMaker supports many script steps in Custom Web Publishing. See "FileMaker scripts and Custom Web Publishing" on page 18.
- You can pass a parameter value to a FileMaker script. See "–script.param (Pass parameter to Script) query parameter" on page 60, "–script.prefind.param (Pass parameter to Script before Find) query parameter" on page 60, and "–script.presort.param (Pass parameter to Script before Sort) query parameter" on page 61.
- The `fmresultset` XML grammar enables you to access fields by name and manipulate `relatedset` (portal) data.
- To access data in a database, you must specify a layout. See chapter 5, "Valid names used in XML query strings,"

Web publishing requirements

What is required to publish a database using Custom Web Publishing

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

- a FileMaker Server deployment that includes:
  - a web server, either Microsoft IIS (Windows) or Apache (macOS)
  - the FileMaker Database Server, enabled for Custom Web Publishing
  - the Web Publishing Engine, installed and configured
- 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

See FileMaker Server Installation and Configuration Guide.

What web users need to access a Custom Web Publishing solution

To access a Custom Web Publishing solution that uses XML, 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.

Where to go from here

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

- If you haven't already done so, use FileMaker Server Admin Console to enable Custom Web Publishing. See FileMaker Server Help and FileMaker Server Installation and Configuration 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 in a database" on page 13.
- To learn how to access data in FileMaker databases using XML, see "Accessing XML data via the Web Publishing Engine" on page 28.

Chapter 4
Accessing XML data with the Web Publishing Engine

You can obtain and update FileMaker data in Extensible Markup Language (XML) format by using the Web Publishing Engine. Many individuals, organizations, and businesses use XML to transfer product information, transactions, inventory data, and other business data.

Using Custom Web Publishing with XML

If you know standard XML, you can start using the Web Publishing Engine after learning a few unique details about Custom Web Publishing with XML, such as the URL syntax and query parameters to use.

By using HTTP URL requests with FileMaker query commands and parameters, you can query a database hosted by FileMaker Server and download the resulting data in XML format. For example, you can query a database for all records in a certain postal code, and use the resulting XML data in whatever way you want to.

See the FileMaker Knowledge Base.

Note The Web Publishing Engine generates XML data that is well-formed and compliant with the XML 1.0 specification. For details about the requirements for well-formed XML, see the XML specification at www.w3.org.

Differences between the Web Publishing Engine and FileMaker Pro XML Import/Export

The Web Publishing Engine and FileMaker Pro both enable you to use XML data with FileMaker databases. There are, however, some important differences between the two methods:

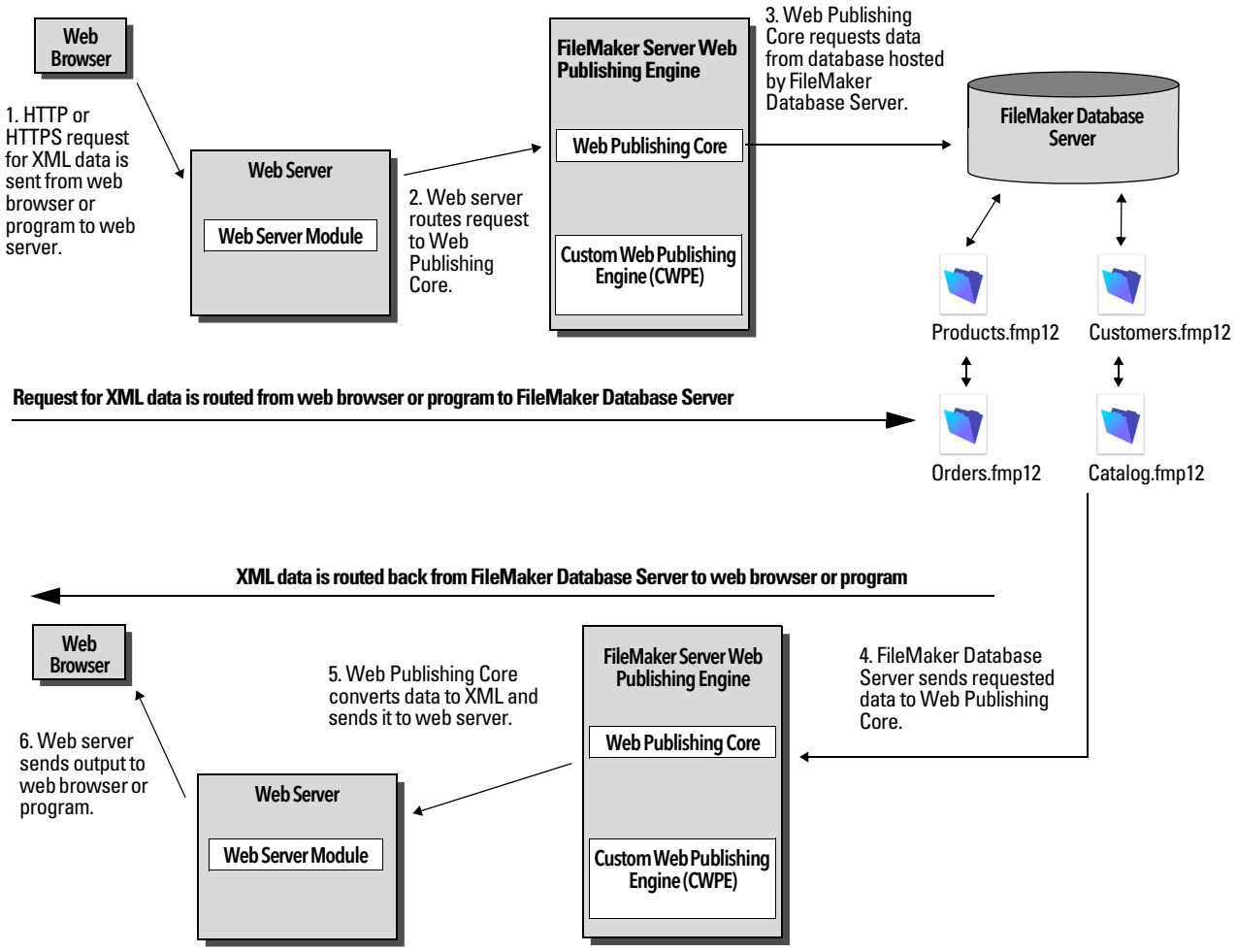
- For accessing XML data, the Web Publishing Engine supports the `fmresultset`, `FMPXMLRESULT`, and `FMPXMLLAYOUT` grammars. For XML import, FileMaker Pro uses the `FMPXMLRESULT` grammar, and for export, FileMaker Pro uses the `FMPXMLRESULT` grammar. See “Accessing XML data via the Web Publishing Engine” on page 28.
- To access XML data with the Web Publishing Engine, you use a Web Publishing Engine query string in a URL. To import and export XML with FileMaker Pro, you use FileMaker Pro menu commands or scripts.
- The Web Publishing Engine is server-based and can be installed on the same or a different host than FileMaker Server. FileMaker Pro XML import and export is desktop-based.
- You can dynamically access XML data from FileMaker databases by using URL requests with the Web Publishing Engine. The FileMaker Pro XML export feature generates a pre-specified XML data file.
- Working with XML data via the Web Publishing Engine is an interactive operation. FileMaker Pro XML import and export is a batch operation.
- The Web Publishing Engine can access XML data from a FileMaker portal, but FileMaker Pro cannot.

- The Web Publishing Engine can access data in a container field, but FileMaker Pro cannot.
- The Web Publishing Engine provides real-time access to FileMaker data via HTTP or HTTPS, but FileMaker Pro cannot.

Note For information on using FileMaker Pro to import and export data in XML format, see FileMaker Pro Help.

How the Web Publishing Engine generates XML data from a request

After a request for XML data is sent to the web server, the Web Publishing Engine queries the FileMaker database and returns the data as an XML document.



General process for accessing XML data from the Web Publishing Engine

Here is an overview of the process for using the Web Publishing Engine to access XML data in a FileMaker database:

1. In the FileMaker Server Admin Console, make sure XML Publishing is enabled. See FileMaker Server Help.
2. In FileMaker Pro, open each FileMaker database that you're publishing and make sure the database has the fmxml extended privilege enabled for XML Custom Web Publishing. See "Enabling Custom Web Publishing in a database" on page 13.

   To access XML data in a portal, set the view for the database layout to **View as Form** or **View as List**. If a user or script changes the view of the database layout to **View as Table**, only the first related record (first row of the portal) is accessible as XML data.

   The XML data is output in an order that corresponds to the order in which field objects were added to the layout. If you want the XML data order to match the order in which fields appear on the screen (top-to-bottom, left-to-right), then select all fields, group them, and then ungroup them. This procedure resets the layout order to match the screen order.
3. Send an HTTP or HTTPS request in the form of a URL that specifies the FileMaker XML grammar, one query command, and one or more FileMaker query parameters to the Web Publishing Engine through an HTML form, an HREF link, or a script in your program or webpage. You can also type the URL in a web browser.

   For information on specifying the URL, see the next section, "About the URL syntax for XML data and container objects." For information on query commands and parameters, see "Using FileMaker query strings to request XML data" on page 38, and chapter 5, "Valid names used in XML query strings."
4. The Web Publishing Engine uses the grammar you specified in the URL to generate XML data containing the results of your request, such as a set of records from the database, and returns it to your program or web browser.
5. The web browser, if it has an XML parser, displays the data, or the program uses the data in the way you specified.

About the URL syntax for XML data and container objects

This section describes the URL syntax for using the Web Publishing Engine to access XML data and container objects from FileMaker databases.

About the URL syntax for XML data

The URL syntax for using the Web Publishing Engine to access XML data from FileMaker databases is:

```
<scheme>://<host>[:<port>]/fmi/xml/<xml_grammar>.xml[?<query string>]
```

where:

- `<scheme>` can be the HTTP or HTTPS protocol.
- `<host>` is the IP address or domain name of the host where the web server is installed.
- `<port>` is optional and specifies the port that the web server is using. If no port is specified, then the default port for the protocol is used (port 80 for HTTP, or port 443 for HTTPS).

- `<xml_grammar>` is the name of the FileMaker XML grammar. Possible values are `fmresultset`, `FMPXMLRESULT`, or `FMPXMLLAYOUT`. See “Using the fmresultset grammar” on page 29 and “Using other FileMaker XML grammars” on page 33.
- `<query string>` is a combination of one query command and one or more query parameters for FileMaker XML publishing. (The `-dbnames` command doesn't require any parameters.) See “Using FileMaker query strings to request XML data” on page 38, and chapter 5, “Valid names used in XML query strings.”

Note The URL syntax, including the names of the query command and parameters, is case sensitive except for portions of the query string. The majority of the URL is in lowercase, with the exception of the two uppercase grammar names: `FMPXMLRESULT` and `FMPXMLLAYOUT`. For information on the rules for case sensitivity of the query string, see “Guidelines for using query commands and parameters” on page 43.

Examples

```
http://server.company.com/fmi/xml/fmresultset.xml?-db=products&-lay=sales
&-findall
http://192.168.123.101/fmi/xml/FMPXMLRESULT.xml?-db=products&-lay=sales
&-findall
```

About the URL syntax for FileMaker container objects in XML solutions

In a generated XML document for an XML solution, the syntax used to refer to a container object is different for container fields that store the actual object in the database, as opposed to container fields that store a reference to the object.

If a container field stores the actual object in the database

The container field's `<data>` element uses the following relative URL syntax to refer to the object:

```
<data>/fmi/xml/cnt/data.<extension>?<query string></data>
```

where `<extension>` is the filename extension identifying the type of object, such as .jpg. The filename extension sets the MIME type to allow the web browser to properly identify the container data. For information on `<query string>`, see the previous section, “About the URL syntax for XML data.”

Example

```
<data>/fmi/xml/cnt/data.jpg?-db=products&-lay=sales
&-field=product_image(1)&-recid=2</data>
```

Note In the generated XML for a container field, the value for the `-field` query parameter is a fully qualified field name. The number in the parentheses indicates the repetition number for the container field, and is generated for both repeating and non-repeating fields. See “About the syntax for a fully qualified field name” on page 45.

To retrieve the container data from the database, use the following syntax:

```
<scheme>://<host>[:<port>]/fmi/xml/cnt/data.<extension>?<query string>
```

For information about <scheme>, <host>, or <port>, see the previous section, “About the URL syntax for XML data.”

Example

```
http://www.company.com/fmi/xml/cnt/data.jpg?-db=products&-lay=sales
&-field=product_image(1)&-recid=2
```

If a container field stores a file reference instead of an actual object

The container field’s <data> element contains a relative path that refers to the object.

Example

```
<data>/images/logo.jpg</data>
```

Note The referenced container object must be stored in the FileMaker Pro Web folder when the record is created or edited, and then copied or moved to a folder with the same relative location in the root folder of the web server software. See “About publishing the contents of container fields on the web” on page 15.

If a container field is empty

The container field’s <data> element is empty.

About URL text encoding

The URLs for accessing XML data and container objects must be encoded in UTF-8 (Unicode Transformation 8 Bit) format. See “About UTF-8 encoded data” on page 38.

Example

To set the value of the “info” field to fiancée, you could use the following URL:

```
http://server.company.com/fmi/xml/fmresultset.xml?-db=members
&-lay=relationships&-recid=2&info=fianc%C3%A9e&-edit
```

%C3%A9 is the URL encoded UTF-8 representation of the é character.

See the URL specification at www.w3.org.

Accessing XML data via the Web Publishing Engine

To access XML data via the Web Publishing Engine, you use a URL that specifies the name of the FileMaker grammar to use, one FileMaker query command, and one or more FileMaker query parameters. The Web Publishing Engine generates XML data from your database that is formatted by one of the following types of XML grammars:

- **fmresultset**: This is the recommended grammar for the Web Publishing Engine for accessing XML data. It is flexible and is optimized for easier field access by name and for easier manipulation of `relatedset` (portal) data. This grammar is also more directly linked to FileMaker terminology and features such as global storage options and identification of summary and calculation fields. To facilitate web publishing, this grammar is designed to be more verbose than the `FMPXMLRESULT` grammar. See “Using the fmresultset grammar” on page 29.
- **FMPXMLRESULT and FMPXMLLAYOUT:** You can also use the `FMPXMLRESULT` and `FMPXMLLAYOUT` grammars with the Web Publishing Engine for accessing XML data. To use one stylesheet for both XML export and Custom Web Publishing, you must use the `FMPXMLRESULT` grammar. To access value lists and field display information in layouts, you must use the `FMPXMLLAYOUT` grammar. See “Using other FileMaker XML grammars” on page 33.

Depending on the grammar you specify in the URL request, the Web Publishing Engine will generate an XML document using one of the grammars. Each XML document contains a default XML namespace declaration for the grammar. See the next section, “About namespaces for FileMaker XML.” Use one of these grammars in your document or webpage to display and work with FileMaker data in XML format.

Note XML data generated by the Web Publishing Engine is encoded using UTF-8 format (Unicode Transformation Format 8). See “About UTF-8 encoded data” on page 38.

About namespaces for FileMaker XML

Unique XML namespaces help distinguish XML tags by the application they were designed for. For example, if your XML document contains two `<DATABASE>` elements, one for FileMaker XML data and another for Oracle XML data, the namespaces will identify the `<DATABASE>` element for each.

The Web Publishing Engine generates a default namespace for each grammar.

For this grammar	This default namespace is generated
`fmresultset`	`xmlns="http://www.filemaker.com/xml/fmresultset"`
`FMPXMLRESULT`	`xmlns="http://www.filemaker.com/fmpxmlresult"`
`FMPXMLLAYOUT`	`xmlns="http://www.filemaker.com/fmpxmllayout"`

About FileMaker database error codes

The Web Publishing Engine returns an error code in the error code elements at the beginning of each XML document that represents the error, if any, in the execution of the most recently executed query command. A value of zero (0) is returned for no error.

For this grammar	This syntax is used
`fmresultset`	`<error code="0"></error>`
`FMPXMLRESULT`	`<ERRORCODE>0</ERRORCODE>`
`FMPXMLLAYOUT`	`<ERRORCODE>0</ERRORCODE>`

The error code element in the XML document indicates errors related to the database and query strings. See appendix A, “Error codes for Custom Web Publishing.”

Retrieving the document type definitions for the FileMaker grammars

You can retrieve the document type definitions (DTDs) for the FileMaker grammars by using an HTTP request.

For this grammar	Use this HTTP request
`fmresultset`	`http://<host>[:<port]/fmi/xml/fmresultset.dtd`
`FMPXMLRESULT`	`http://<host>[:<port]/fmi/xml/FMPXMLRESULT.dtd`
`FMPXMLLAYOUT`	`http://<host>[:<port]/fmi/xml/FMPXMLLAYOUT.dtd`

Using the fmresultset grammar

The XML element names in this grammar use FileMaker terminology, and the storage of fields is separated from the type of fields. The grammar also includes the ability to identify summary, calculation, and global fields.

To use the `fmresultset` grammar, specify the following name of the `fmresultset` grammar in the URL requesting the XML document from the Web Publishing Engine:

`fmresultset.xml`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-findall
```

Note When specifying the `fmresultset` grammar, be sure to use lowercase.

The Web Publishing Engine will generate an XML document using the `fmresultset` grammar. In the XML document, the Web Publishing Engine will reference the document type definition for the `fmresultset` grammar in the `<!DOCTYPE>` instruction in the second line of the document, immediately after the `<?xml . . . ?>` instruction. The `<!DOCTYPE>` instruction specifies the URL for downloading the DTD for the `fmresultset` grammar.

Description of elements in the fmresultset grammar

The `fmresultset` grammar consists primarily of the `<datasource>` element, the `<metadata>` element, and the `<resultset>` element.

<datasource> element

In the `fmresultset` grammar, the `<datasource>` element contains the table, layout, date-format, time-format, timestamp-format, total-count, and database attributes.

- The date-format attribute of the `<datasource>` element specifies the format of dates in the XML document:

  `MM/dd/yyyy`

where:

  - `MM` is the 2-digit value for the month (01 through 12, where 01 is January and 12 is December)
  - `dd` is the 2-digit value for the day of the month (01 through 31)
  - `yyyy` is the 4-digit value for the year

- The time-format attribute of the `<datasource>` element specifies the format of times in the XML document:

  `HH:mm:ss`

where:

  - `HH` is the 2-digit value for hours (00 through 23, for the 24-hour format)
  - `mm` is the 2-digit value for minutes (00 through 59)
  - `ss` is the 2-digit value for seconds (00 through 59)

- The timestamp-format attribute of the `<datasource>` element combines the formats of date-format and time-format into one timestamp:

  `MM/dd/yyyy HH:mm:ss`

<metadata> element

The `<metadata>` element of the `fmresultset` grammar contains one or more `<field-definition>` and `<relatedset-definition>` elements, each containing attributes for one of the fields of the result set.

The `<field-definition>` attributes specify:

- whether the field is an `auto-enter` field (`"yes"` or `"no"`)
- whether the field is a `four-digit-year` field (`"yes"` or `"no"`)
- whether it is a global field (`"yes"` or `"no"`)
- the maximum number of repeating values (`max-repeat` attribute)
- the maximum number of characters allowed (`max-characters` attribute)
- whether it is a `not-empty` field (`"yes"` or `"no"`)
- whether it is for numeric data only (`"yes"` or `"no"`)
- result (`"text"`, `"number"`, `"date"`, `"time"`, `"timestamp"`, or `"container"`)
- whether it is a `time-of-day` field (`"yes"` or `"no"`)
- type (`"normal"`, `"calculation"`, or `"summary"`)
- and the field name (fully qualified as necessary)

The `<relatedset-definition>` element represents a portal. Each related field in a portal is represented by the `<field-definition>` element contained within the `<relatedset-definition>` element. If there are multiple related fields in a portal, the field definitions for the related fields are grouped within a single `<relatedset-definition>` element.

<resultset> element

The `<resultset>` element contains the `<record>` elements returned as the result of a query and an attribute for the total number of records found. Each `<record>` element contains the field data for one record in the result set—including the `mod-id` and the `record-id` attributes for the record, and the `<data>` element containing the data for one field in the record.

Each record in a portal is represented by a `<record>` element within the `<relatedset>` element. The count attribute of the `<relatedset>` element specifies the number of records in the portal, and the table attribute specifies the table associated with the portal.

XML data in the fmresultset grammar

Example

```
<fmresultset xmlns="http://www.filemaker.com/xml/fmresultset" version="1.0">
  <error code="0"/>
  <product build="03/29/2017" name="FileMaker Web Publishing Engine"
   version="16.0.1.0"/>
  <datasource database="art" date-format="MM/dd/yyyy" layout="web3"
   table="art" time-format="HH:mm:ss" timestamp-format="MM/dd/yyyy HH:mm:ss"
   total-count="12"/>
  <metadata>
    <field-definition auto-enter="no" four-digit-year="no" global="no" max-
     repeat="1" name="Title" not-empty="no" numeric-only="no" result="text"
     time-of-day="no" type="normal"/>
    <field-definition auto-enter="no" four-digit-year="no" global="no" max-
     repeat="1" name="Artist" not-empty="no" numeric-only="no" result="text"
     time-of-day="no" type="normal"/>
    <field-definition auto-enter="no" four-digit-year="no" global="no" max-
     repeat="1" name="Style" not-empty="no" numeric-only="no" result="text"
     time-of-day="no" type="normal"/>
    <field-definition auto-enter="no" four-digit-year="no" global="no" max-
     repeat="1" name="length" not-empty="no" numeric-only="no"
     result="number" time-of-day="no" type="calculation"/>
    <relatedset-definition table="artlocations">
      <field-definition auto-enter="no" four-digit-year="no" global="no" max-
       repeat="1" name="artlocations::Location" not-empty="no" numeric-
       only="no" result="text" time-of-day="no" type="normal"/>
      <field-definition auto-enter="no" four-digit-year="no" global="no" max-
       repeat="1" name="artlocations::Date" not-empty="no" numeric-only="no"
       result="date" time-of-day="no" type="normal"/>
    </relatedset-definition>
  </metadata>
  <resultset count="1" fetch-size="1">
    <record mod-id="6" record-id="14">
      <field name="Title">
        <data>Spring in Giverny 3</data>
      </field>
      <field name="Artist">
        <data>Claude Monet</data>
      </field>
      <field name="Style">
        <data/>
      </field>
      <field name="length">
        <data>19</data>
      </field>
      <relatedset count="0" table="artlocations"/>
    </record>
  </resultset>
</fmresultset>
```

Using other FileMaker XML grammars

The other FileMaker XML grammars contain information about field types, value lists, and layouts. FMPXMLRESULT is functionally equivalent to fmresultset. To access value lists and field display information in layouts, you must use the FMPXMLLAYOUT grammar. The FMPXMLRESULT and FMPXMLLAYOUT grammars are more compact for data interchange.

To use the FMPXMLRESULT grammar, specify the following grammar name in the URL requesting the XML document from the Web Publishing Engine:

```
FMPXMLRESULT.xml
```

Example

```
http://192.168.123.101/fmi/xml/FMPXMLRESULT.xml?-db=employees&-lay=family
&-findall
```

To use the FMPXMLLAYOUT grammar, specify the following grammar name with the -view query command in the URL requesting the XML document from the Web Publishing Engine:

```
FMPXMLLAYOUT.xml
```

Example

```
http://192.168.123.101/fmi/xml/FMPXMLLAYOUT.xml?-db=employees&-lay=family
&-view
```

Note When specifying the FMPXMLRESULT and FMPXMLLAYOUT grammars, be sure to enter the grammar name in uppercase.

In the generated XML document, the Web Publishing Engine will reference the document type definition for the grammar in the <!DOCTYPE> instruction in the second line of the document, immediately after the <?xml . . .?> instruction. The <!DOCTYPE> instruction specifies the URL for downloading the DTD for the grammar.

Description of elements in the FMPXMLRESULT grammar

In the FMPXMLRESULT grammar, the <DATABASE> element contains the NAME, RECORDS, DATEFORMAT, LAYOUT, and TIMEFORMAT attributes.

The DATEFORMAT attribute of the <DATABASE> element specifies the format of dates in the XML document. The TIMEFORMAT attribute of the <DATABASE> element specifies the format of times in the XML document. The date and time formats for the FMPXMLRESULT and the fmresultset grammars are the same. See “Description of elements in the fmresultset grammar” on page 30.

The <METADATA> element of the FMPXMLRESULT grammar contains one or more <FIELD> elements, each containing information for one of the fields/columns of the result set—including the name of the field as defined in the database, the field type, the Yes or No allowance for empty fields (EMPTYOK attribute) and the maximum number of repeating values (MAXREPEAT attribute). Valid values for field types are TEXT, NUMBER, DATE, TIME, TIMESTAMP, and CONTAINER.

The <RESULTSET> element contains all of the <ROW> elements returned as the result of a query and an attribute for the total number of records found. Each <ROW> element contains the field/column data for one row in the result set. This data includes the RECORDID and MODID for the row (see “–modid (Modification ID) query parameter” on page 56), and the <COL> element. The <COL> element contains the data for one field/column in the row where multiple <DATA> elements represent one of the values in a repeating or portal field.

XML data in the FMPXMLRESULT grammar

Example

```
<FMPXMLRESULT xmlns="http://www.filemaker.com/fmpxmlresult">
  <ERRORCODE>0</ERRORCODE>
  <PRODUCT BUILD="03/29/2017" NAME="FileMaker Web Publishing Engine"
   VERSION="16.0.1.0"/>
  <DATABASE DATEFORMAT="MM/dd/yyyy" LAYOUT="web" NAME="art" RECORDS="12"
   TIMEFORMAT="HH:mm:ss"/>
  <METADATA>
    <FIELD EMPTYOK="YES" MAXREPEAT="1" NAME="Title" TYPE="TEXT"/>
    <FIELD EMPTYOK="YES" MAXREPEAT="1" NAME="Artist" TYPE="TEXT"/>
    <FIELD EMPTYOK="YES" MAXREPEAT="1" NAME="Image" TYPE="CONTAINER"/>
  </METADATA>
  <RESULTSET FOUND="1">
    <ROW MODID="7" RECORDID="4">
      <COL>
        <DATA>Village Market</DATA>
      </COL>
      <COL>
        <DATA>Camille Pissarro</DATA>
      </COL>
      <COL>
        <DATA>/fmi/xml/cnt/Untitled.pct?-db=art&-lay=web&-recid=4
         &-field=Image(1)
        </DATA>
      </COL>
    </ROW>
  </RESULTSET>
</FMPXMLRESULT>
```

The order of the <COL> elements corresponds with the order of the <FIELD> elements in the <METADATA> element—for example, where the “Title” and “Artist” fields are listed in the <METADATA> element, “Village Market” and then “Camille Pissarro” are listed in the same order in the <RESULTSET> and <ROW> elements.

Description of elements in the FMPXMLLAYOUT grammar

In the FMPXMLLAYOUT grammar, the <LAYOUT> element contains the name of the layout, the name of the database, and <FIELD> elements for each field found in the corresponding layout in the database. Each <FIELD> element describes the style type of the field, and contains the VALUELIST attribute for any associated value list of the field.

The <VALUELISTS> element contains one or more <VALUELIST> elements for each value list found in the layout—each including the name of the value list and a <VALUE> element for each value in the list.

Depending on the options selected in the **Specify Fields for Value List** dialog box in the FileMaker database, the <VALUE> element contains a DISPLAY attribute that contains the value in the first field only, the second field only, or both fields of a value list. For example, suppose the first field in a value list stores the art style's ID number (such as "100"), and the second field displays the art style's associated name (such as "Impressionism"). Here is a summary of the contents of the DISPLAY attribute when the various combinations of options are selected in the **Specify Fields for Value List** dialog box:

- If **Also display values from second field** is not selected, the DISPLAY attribute contains the value in the first field of a value list only.

Example

The DISPLAY attribute contains the art style's ID number only:

```
<VALUELISTS>
  <VALUELIST NAME="style">
    <VALUE DISPLAY="100">100</VALUE>
    <VALUE DISPLAY="101">101</VALUE>
    <VALUE DISPLAY="102">102</VALUE>
  </VALUELIST>
</VALUELISTS>
```

- If **Also display values from second field** and **Show values only from second field** are both selected, the DISPLAY attribute contains the value in the second field only.

Example

The DISPLAY attribute contains the art style's name only:

```
<VALUELISTS>
    <VALUELIST NAME="style">
      <VALUE DISPLAY="Impressionism">100</VALUE>
      <VALUE DISPLAY="Cubism">101</VALUE>
      <VALUE DISPLAY="Abstract">102</VALUE>
    </VALUELIST>
</VALUELISTS>
```

- If **Also display values from second field** is selected and **Show values only from second field** is not selected, the DISPLAY attribute contains the values in both fields of a value list.

Example

The DISPLAY attribute contains both the art style's ID number and the art style's name:

```
<VALUELISTS>
  <VALUELIST NAME="style">
      <VALUE DISPLAY="100 Impressionism">100</VALUE>
      <VALUE DISPLAY="101 Cubism">101</VALUE>
      <VALUE DISPLAY="102 Abstract">102</VALUE>
  </VALUELIST>
</VALUELISTS>
```

For date, time, and timestamp fields, data for value lists are formatted using the "fm" format for that field type. The "fm" formats are `MM/DD/YYYY` for date, `hh:mm:ss` for time, and `MM/DD/YYYY hh:mm:ss` for timestamp. For example, if a "birthdays" value list is used for a pop-up menu on a "birthdate" field of a layout, and the "birthdate" field is of type date, then the values output for that value list will all be in the "fm" date format.

Note If two fields with different field types on a layout share the same value list, the first field's type determines the format of the value list data.

XML data in the FMPXMLLAYOUT grammar

Example

```
<FMPXMLLAYOUT xmlns="http://www.filemaker.com/fmpxmllayout">
  <ERRORCODE>0</ERRORCODE>
  <PRODUCT BUILD="03/29/2017" NAME="FileMaker Web Publishing Engine"
   VERSION="16.0.1.0"/>
  <LAYOUT DATABASE="art" NAME="web2">
    <FIELD NAME="Title">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
    <FIELD NAME="Artist">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
    <FIELD NAME="Image">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
    <FIELD NAME="artlocations::Location">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
    <FIELD NAME="artlocations::Date">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
    <FIELD NAME="Style">
      <STYLE TYPE="POPUPMENU" VALUELIST="style"/>
    </FIELD>
  </LAYOUT>
  <VALUELISTS>
    <VALUELIST NAME="style">
      <VALUE DISPLAY="Impressionist">Impressionist</VALUE>
      <VALUE DISPLAY="Modern">Modern</VALUE>
      <VALUE DISPLAY="Abstract">Abstract</VALUE>
    </VALUELIST>
  </VALUELISTS>
</FMPXMLLAYOUT>
```

About UTF-8 encoded data

All XML data generated by the Web Publishing Engine is encoded in UTF-8 (Unicode Transformation 8 Bit) format. This format compresses data from the standard Unicode format of 16 bits to 8 bits for ASCII characters. XML parsers are required to support Unicode and UTF-8 encoding.

UTF-8 encoding includes direct representations of the values of 0-127 for the standard ASCII set of characters used in English, and provides multibyte encodings for Unicode characters with higher values.

Note Be sure to use a web browser or text editor program that supports UTF-8 files.

The UTF-8 encoding format includes the following features:

- All ASCII characters are one-byte UTF-8 characters. A legal ASCII string is a legal UTF-8 string.
- Any non-ASCII character (any character with the high-order bit set) is part of a multibyte character.
- The first byte of any UTF-8 character indicates the number of additional bytes in the character.
- The first byte of a multibyte character is easily distinguished from the subsequent byte, which makes it is easy to locate the start of a character from an arbitrary position in a data stream.
- It is easy to convert between UTF-8 and Unicode.
- The UTF-8 encoding is relatively compact. For text with a large percentage of ASCII characters, it is more compact than Unicode. In the worst case, a UTF-8 string is only 50% larger than the corresponding Unicode string.

Using FileMaker query strings to request XML data

To request XML data from a FileMaker database, you use the FileMaker query commands and parameters in a query string.

Example

You can use the `-findall` query command in the following query string in a URL to request a list of all products in a FileMaker database named “products”:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=products
&-lay=sales&-findall
```

A query string must contain only one query command, such as `-new`. Most query commands also require various matching query parameters in the query string. For example, all query commands except `-dbnames` require the `-db` parameter that specifies the database to query.

You can also use query commands and parameters in a URL.

This section contains a summary of the FileMaker query commands and parameters. For information about using them in a query string, see “Valid names used in XML query strings” on page 42.

Use this query command name	To execute this command
-dbnames	Retrieve names of all hosted and web-shared databases.
-delete	Delete record.
-dup	Duplicate record.
-edit	Edit record.
-find	Find record(s).
-findall	Find all records.
-findany	Find a random record.
-findquery	Perform complex or compound find request.
-layoutnames	Retrieve names of all available layouts for a hosted and web-shared database.
-new	Add new record.
-scriptnames	Retrieve names of all available scripts for a hosted and web-shared database.
-view	Retrieves layout information from a database if the FMPXMLLAYOUT grammar is specified. Retrieves <metadata> section of XML document and an empty record set if the fmresultset or FMPXMLRESULT grammar is specified.

Use these query parameter names	With these query commands
-db (database name)	Required with all query commands except -dbnames
-delete.related	Optional with -edit
-field	Required to specify a field in a URL for container requests. See “About the URL syntax for FileMaker container objects in XML solutions” on page 26.
fieldname	At least one field name is required with -edit. Optional with -find. See “fieldname (Non-container field name) query parameter” on page 52.
fieldname.op (operator)	Optional with -find
-lay (layout name)	Required with all query commands, except -dbnames, -layoutnames, and -scriptnames
-lay.response (switch layout for XML response)	Optional with all query commands, except -dbnames, -layoutnames, and -scriptnames
-lop (logical operator)	Optional with -find
-max (maximum records)	Optional with -find, -findall, and -findquery
-modid (modification ID)	Optional with -edit
-query	Required with -findquery compound find requests
-recid (record ID)	Required with -edit, -delete, -dup. Optional with -find
-relatedsets.filter	Optional with -find, -findall, -findany, -edit, -new, -dup, and -findquery
-relatedsets.max	Optional with -find, -edit, -new, -dup, and -findquery
-script (perform script)	Optional with -find, -findall, -findany, -new, -edit, -delete, -dup, -view, and -findquery

Use these query parameter names	With these query commands
`-script.param` (pass a parameter value to the script specified by `-script`)	Optional with `-script` and `-findquery`
`-script.prefind` (perform script before `-find`, `-findany`, and `-findall`)	Optional with `-find`, `-findany`, `-findall`, and `-findquery`
`-script.prefind.param` (pass a parameter value to the script specified by `-script.prefind`)	Optional with `-script.prefind` and `-findquery`
`-script.presort` (perform script before sort)	Optional with `-find`, `-findall`, and `-findquery`
`-script.presort.param` (pass a parameter value to the script specified by `-script.presort`)	Optional with `-script.presort` and `-findquery`
`-skip` (skip records)	Optional with `-find`, `-findall`, and `-findquery`
`-sortfield.[1-9]` (sort field)	Optional with `-find`, `-findall`, and `-findquery`
`-sortorder.[1-9]` (sort order)	Optional with `-find`, `-findall`

Switching layouts for an XML response

The `-lay` query parameter specifies the layout you want to use when requesting XML data. Often, the same layout is appropriate for processing the data that results from the request. In some cases, you might want to search for data using a layout which contains fields that, for security reasons, don't exist in another layout you want to use for displaying the results. (To do a search for data in a field, the field must be placed on the layout you specify in the XML request.)

To specify a different layout for displaying an XML response than the layout used for processing the XML request, you can use the optional `-lay.response` query parameter.

Example

The following request searches for values greater than 100,000 in the "Salary" field on the "Budget" layout. The resulting data is displayed using the "ExecList" layout, which does not include the "Salary" field.

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=Budget&Salary=100000&Salary.op=gt&-find&-lay.response=ExecList
```

Understanding how an XML request is processed

There are several query parameters that affect the processing of an XML request and the generation of an XML document.

Here is the order in which FileMaker Server and the Web Publishing Engine process an XML request:

1. Process the `-lay` query parameter.
2. Set the global field values specified in the query (the "`.global=`" portion of a URL).
3. Process the `-script.prefind` query parameter, if specified.
4. Process the query commands, such as `-find` or `-new`.
5. Process the `-script.presort` query parameter, if specified.
6. Sort the resulting data, if a sort was specified.
7. Process the `-script` query parameter, if specified.
8. Process the `-lay.response` query parameter to switch to a different layout, if this is specified.
9. Generate the XML document.

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

For example, consider a request that deletes the current record, sorts the records, and then executes a script. If the `-sortfield` parameter specifies a nonexistent field, the request deletes the current record and returns error code 102 ("Field is missing"), but does not execute the script.

Troubleshooting XML document access

If you have trouble accessing XML documents with the Web Publishing Engine, verify that:

- The extended privileges in the database are set for XML Custom Web Publishing and assigned to a user account. See "Enabling Custom Web Publishing in a database" on page 13.
- The database is hosted on the Database Server component of the FileMaker Server deployment, and is opened by FileMaker Server. See FileMaker Server Help.
- The database account name and password you are using, if any, are correct.
- The web server component of the FileMaker Server deployment is running.
- The Web Publishing Engine component of the FileMaker Server deployment is running.
- XML Publishing is enabled in the Web Publishing Engine component. See FileMaker Server Help.

Chapter 5
Valid names used in XML query strings

This chapter describes the valid names of query commands and parameters you can use in an XML query string when accessing FileMaker data using the Web Publishing Engine.

About the query commands and parameters

The following is a complete list of the query command names and query parameter names:

Query command names	Query parameter names
`-dbnames` (See page 47.)	`-db` (See page 51.)
`-delete` (See page 47.)	`-field` (See page 52.)
`-dup` (See page 48.)	`fieldname` (See page 52.)
`-edit` (See page 48.)	`fieldname.op` (See page 53.)
`-find`, `-findall`, `-findany` (See page 49.)	`-lay` (See page 54.)
`-findquery` (See page 49.)	`-lay.response` (See page 54.)
`-layoutnames` (See page 50.)	`-lop` (See page 55.)
`-new` (See page 50.)	`-max` (See page 55.)
`-scriptnames` (See page 51.)	`-modid` (See page 56.)
`-view` (See page 51.)	`-query` (See page 56.)
	`-recid` (See page 57.)
	`-relatedsets.filter` (See page 58.)
	`-relatedsets.max` (See page 59.)
	`-script` (See page 59.)
	`-script.param` (See page 60.)
	`-script.prefind` (See page 60.)
	`-script.prefind.param` (See page 60.)
	`-script.presort` (See page 61.)
	`-script.presort.param` (See page 61.)
	`-skip` (See page 62.)
	`-sortfield.[1-9]` (See page 62.)
	`-sortorder.[1-9]` (See page 63.)

Important The `-lay` parameter for specifying a layout is required with all query commands except `-dbnames`, `-layoutnames`, and `-scriptnames`.

Guidelines for using query commands and parameters

When using query commands and parameters in a query string, keep the following guidelines in mind:

- A query string must contain only one query command; no more and no less. For example, a query string can contain `-new` to add a new record, but it can't contain `-new` and `-edit` in the same query string.
- Most query commands require various matching query parameters in the query string. For example, all query commands except `-dbnames` require the `-db` parameter that specifies the database to query. See the table of required parameters in "Using FileMaker query strings to request XML data" on page 38.
- For query parameters and field names, specify the particular value you want to use, such as `-db=employees`. For query commands, don't specify an "=" sign or a value after the command name, such as `-findall`.
- The Web Publishing Engine converts all reserved words to lowercase, including query commands, query parameters, and command values where specific values are expected (for example: `-lop=and`, `-lop=or`, `-sortorder=ascend`, `-sortorder=descend`, `-max=all`).
- Database names, layout names, and field names used in query strings are case insensitive, such as using `-lay=mylayout` to specify the layout name `MyLayout`.
- It is not recommended to use periods or parentheses in field names. In some cases, field names with periods may work, but field names with the following exceptions can never be used:
  - The period cannot be followed by a number. For example, `myfield.9` is an invalid field name.
  - The period cannot be followed by the text string `op` (the two letters "op"). For example, `myfield.op` is an invalid field name.
  - The period cannot be followed by the text string `global` (the word "global"). For example, `myfield.global` is an invalid field name.

    Field names containing any of these exceptions cannot be accessed via XML using an HTTP query. These constructs are reserved for record IDs, as described in the section, "About the syntax for a fully qualified field name" on page 45.
- For the `-find` command, the value of a field is case insensitive. For example, you can use `Field1=Blue` or `Field1=blue`. For the `-new` and `-edit` commands, the case you use in the value of a field is preserved and stored in the database exactly as you specify in the query string. For example, `LastName=Doe`.

Query command parsing

The Web Publishing Engine parses query commands in the following order, and stops parsing XML queries with the first error. If an error code is returned, the error code returned matches the first error that is identified.

1. Does the query have a command and is the query command valid?

   It is an error if the query is missing the command or uses an unknown command.

   Example

   ```
   -database
   ```

2. Does the query have two commands?

   Example

   ```
   -find&-edit
   ```

3. Does the query have an invalid value for a command or parameter?

   Example

   ```
   -lop=amd
   ```

4. Is the query missing the required database name parameter (`-db` parameter)?
5. Is the query missing the required layout name parameter (`-lay` parameter)?
6. Does the query have an invalid sort?
7. Does the query have invalid field parameters?

Note If a query contains valid but extraneous information, the query is processed without an error. For example, if you specify the `-lop` parameter on a `-delete` command, the `-lop` parameter is ignored because it does not cause the query to be invalid or ambiguous.

For information about specific error codes returned, see appendix A, “Error codes for Custom Web Publishing.”

About the syntax for a fully qualified field name

A fully qualified field name identifies an exact instance of a field. Because fields with common names can be based on different tables, you must use fully qualified names, in some cases, to avoid errors.

The syntax for specifying a fully qualified field name is:

`table-name::field-name(repetition-number).record-id`

where:

- `table-name` is the name of the table that contains the field. The table name is only required if the field is not in the underlying table of the layout specified in the query string.
- `field-name(repetition-number)` is the specific value in a repeating field, and is only required for repeating fields. The repetition number starts counting at the numeral 1. For example, `field-name(2)` refers to the second value in the repeating field. If you don't specify a repetition number for a repeating field, the first value in the repeating field is used. The repetition-number is required for the `-new` and `-edit` query commands involving repeating fields, but it is not required for the `-find` command.
- `record-id` is the record ID, and is only required if you are using a query string to add or edit records in portal fields. See the following sections "Adding records to a portal," and "Editing records in a portal." The `record-id` is required for the `-new` and `-edit` query commands involving portal fields, but it is not required for the `-find` command.

Note To be accessible, fields must be placed on the layout you specify in the query string.

Using query commands with portal fields

The following sections describe how query commands work with portal fields.

Adding records to a portal

To add a new record to a portal at the same time you add a parent record, use the `-new` query command and do the following in query string for the request:

- Use the fully qualified field name for the related portal field.
- Specify 0 as the record ID after the name of the related portal field.
- Specify at least one of the fields for the parent record before specifying the related portal field.
- Specify the data for the match field (key field) in the parent record.

Example

The following URL adds a new parent Employee record for John Doe, and a new related record for Jane in the portal at the same time. The name of the related table is Dependents, and the name of the related field in the portal is Names. The match field, ID, stores an employee ID number.

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&FirstName=John&LastName=Doe&ID=9756&Dependents::Names.0=Jane&-new
```

Note You can only add one related record to a portal per request.

Editing records in a portal

To edit one or more records in a portal, use the `-edit` command and a record ID to specify the parent record that contains the portal records you want to edit. Specify the particular portal record to edit by using its record ID in a fully qualified field name. You can determine a record ID from the record ID attribute of the `<record>` element in the `<relatedset>` element in the XML data. See “Using the fmresultset grammar” on page 29.

Examples

The following URL edits a record in a portal where the parent record has the record ID of 1001. Dependents is the name of the related table, Names is the name of the related field in the portal, and the 2 in Names.2 is the record ID of a portal record.

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-recid=1001&Dependents::Names.2=Kevin&-edit
```

How to use one request to edit multiple portal records via the parent record:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-recid=1001&Dependents::Names.2=Kevin&Dependents::Names.5=Susan&-edit
```

You can also use the `-edit` command and specify 0 as the portal record ID to add a new related record in the portal for an existing parent record.

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-recid=1001&Dependents::Names.0=Timothy&-edit
```

Deleting portal records

To delete portal records, use the `-delete.related` parameter with the `-edit` command rather than using the `-delete` command.

Examples

The following URL deletes record “1001” from the table “employees”:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-recid=1001&-delete
```

But the following URL deletes a portal record with a record ID of “3” from the related table called “Dependents”, with the parent record ID of “1001”.

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-recid=1001&-delete.related=Dependents.3&-edit
```

See “–delete.related (Portal records delete) query parameter” on page 52.

Querying portal fields

In a solution that has many related records, querying and sorting portal records can be time consuming. To restrict the number of records and rows to display in a related set, use the `-relatedsets.filter` and `-relatedsets.max` parameters with find requests. See “–relatedsets.filter (Filter portal records) query parameter” on page 58 and “–relatedsets.max (Limit portal records) query parameter” on page 59.

About the syntax for specifying a global field

The syntax for specifying a global field is:

`table-name::field-name(repetition-number).global`

where `global` identifies a field as using global storage. For information about `table-name` and `field-name(repetition-number)`, see “About the syntax for a fully qualified field name” on page 45. For information on global fields, see FileMaker Pro Help.

You must use the `.global` syntax to identify a global field in a query string. The Web Publishing Engine sets the parameter values for global fields before performing the query command or setting any other parameter values in the query string. For direct XML requests, the global values expire immediately after the request is made.

If you don’t use the `.global` syntax to identify a global field in a query string, the Web Publishing Engine evaluates the global field along with the remainder of the query string without setting the global field value first.

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&Country.global=USA&-recid=1&-edit
```

Query command reference

This section contains information about the query commands available for XML requests.

-dbnames (Database names) query command

Retrieves the names of all databases that are hosted by FileMaker Server and enabled for Custom Web Publishing with XML.

Required query parameters: (none)

Example

To retrieve the database names:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-dbnames
```

-delete (Delete record) query command

Deletes the record specified by the `-recid` parameter.

Required query parameters: `-db`, `-lay`, `-recid`

Optional query parameter: `-script`

Example

To delete a record:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-recid=4&-delete
```

-dup (Duplicate record) query command

Duplicates the record specified by the `-recid` parameter.

Required query parameters: `-db`, `-lay`, `-recid`

Optional query parameter: `-script`

Example

To duplicate the specified record:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-recid=14&-dup
```

-edit (Edit record) query command

Updates the record specified by the `-recid` parameter, populating the fields with the contents of any field name/value pairs. The `-recid` parameter indicates which record should be edited.

Required query parameters: `-db`, `-lay`, `-recid`, one or more field name(s)

Optional query parameter: `-modid`, `-script`, field name

Note The `-edit` command can be used for editing records in a portal. See “Editing records in a portal” on page 46.

Example

To edit a record:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-recid=13&Country=USA&-edit
```

-find, -findall, or -findany (Find records) query commands

Submits a search request using defined criteria.

Required query parameters: `-db`, `-lay`

Optional query parameters: `-recid`, `-lop`, `-op`, `-max`, `-skip`, `-sortorder`, `-sortfield`, `-script`, `-script.prefind`, `-script.presort`, field name

Examples

To find a record by field name:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=family&Country=USA&-find
```

To find a record by record ID:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-recid=427&-find
```

To find all records in the database, use `-findall`:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-findall
```

To find a random record, use `-findany`:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family
&-findany
```

Notes

- Specifying a field name multiple times in a single request is not supported; FileMaker Server parses all of the values, but uses only the last value parsed.
- When using the `-findall` command, avoid computer memory overload problems by specifying a default maximum number of records to return per page by using the `-max` query parameter.

-findquery (Compound find) query command

Submits a search request using multiple find records and omit records requests.

Required query parameters: `-db`, `-lay`, `-query`

Optional query parameters: `-max`, `-skip`, `-sortorder`, `-sortfield`, `-script`, `-script.prefind`, `-script.presort`

Example

Find records for cats or dogs that are not named “Fluffy”:

```
http://host/fmi/xml/fmresultset.xml?-db=vetclinic&-lay=animals
&-query=(q1);(q2);!(q3)&-q1=typeofanimal&-q1.value=Cat&-q2=typeofanimal
&-q2.value=Dog&-q3=name&-q3.value=Fluffy&-findquery
```

Using the -findquery command for compound finds

A `-findquery` statement consists of four parts, in the following order:

- The `-query` parameter.
- The query request declarations, consisting of the query identifier declarations and request operations.
- The search field and value definitions for each query identifier.
  - Define query identifiers. A query identifier is the letter "q" followed by a number. For example: `-q1`
  - Define query identifier values with the parameter. For example: `-q1.value=fieldvalue`
  - Define query identifier operators by including it as part of the `fieldvalue` expression. For example, to use an asterisk as a “begins with” operator: `-q1.value=fieldvalue*`
- The `-findquery` command, at the end of the complete statement.

For information on using the `-query` parameter, see “–query (Compound find request) query parameter” on page 56.

-layoutnames (Layout names) query command

Retrieves the names of all available layouts for a specified database that is hosted by FileMaker Server and enabled for Custom Web Publishing with XML.

Required query parameters: `-db`

Example

To retrieve the names of available layouts:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-layoutnames
```

-new (New record) query command

Creates a new record and populates that record with the contents of any field name/value pairs.

Required query parameters: `-db`, `-lay`

Optional query parameter: one or more field name(s), `-script`

Note For information on including new data for a portal, see “Adding records to a portal” on page 45.

Example

To add a new record:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&Country=Australia&-new
```

-scriptnames (Script names) query command

Retrieves the names of all available scripts for a specified database that is hosted by FileMaker Server and enabled for Custom Web Publishing with XML.

Required query parameters: `-db`

Example

To retrieve the names of all scripts:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-scriptnames
```

-view (View layout information) query command

If the `FMPXMLLAYOUT` grammar is specified, retrieves layout information from a database and displays it in the `FMPXMLLAYOUT` grammar. If a data grammar (`fmresultset` or `FMPXMLRESULT`) is specified, retrieves the metadata section of XML document and an empty record set.

Required query parameters: `-db`, `-lay`

Optional query parameter: `-script`

Examples

To retrieve layout information:

```
http://192.168.123.101/fmi/xml/FMPXMLLAYOUT.xml?-db=employees
&-lay=departments&-view
```

To retrieve metadata information:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-view
```

Query parameter reference

This section contains information about the query parameters available for XML requests.

-db (Database name) query parameter

Specifies the database that the query command is applied to.

Value is: Name of the database, not including the filename extension if any.

Note When specifying the name of the database for the `-db` parameter in query strings, do not include a filename extension. The actual database filename can optionally include an extension, but extensions are not allowed as a value for the `-db` parameter.

Required with: All query commands except `-dbnames`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-findall
```

-delete.related (Portal records delete) query parameter

Deletes a record from a portal field.

Optional with: `-edit` query command

Requires: A related table name and a record id

Example

The following deletes a portal record with a record ID of “20” from the related table called “jobtable”, with a parent record ID of “7”.

```
http://host/fmi/xml/fmresultset.xml?-db=career&-lay=applications&-recid=7
&-delete.related=jobtable.20&-edit
```

-field (Container field name) query parameter

Specifies the name of a container field.

Required with: Requests for data in a container field

See “About the URL syntax for XML data and container objects” on page 25.

fieldname (Non-container field name) query parameter

Field names are used to control criteria for the `-find` query command, or to modify the contents of a record. When you need to specify a value for a non-container field for a query command or parameter, use the field name without the hyphen (`-`) character as the name portion of the name/value pair.

Name is: Name of the field in the FileMaker database. If the field is not in the underlying table of the layout specified in the query string, the field name must be fully qualified.

It is not recommended to use periods or parentheses in field names. In some cases, field names with periods may work, but field names with the following exceptions can never be used:

- The period cannot be followed by a number. For example, `myfield.9` is an invalid field name.
- The period cannot be followed by the text string `op` (the two letters “op”). For example, `myfield.op` is an invalid field name.
- The period cannot be followed by the text string `global` (the word “global”). For example, `myfield.global` is an invalid field name.

Field names containing any of these exceptions cannot be accessed via XML using an HTTP query. Using periods in field names should be reserved for record IDs, as described in “About the syntax for a fully qualified field name” on page 45.

Value is: For the `-new` and `-edit` query commands, specify the value you want to store in the field in the current record. For the `-find` query commands, specify the value you want to search for in the field. When you specify the value for a date, time, or timestamp field, specify the value using the “fm” format for that field type. The “fm” formats are `MM/DD/YYYY` for date, `hh:mm:ss` for time, and `MM/DD/YYYY hh:mm:ss` for timestamp.

Required with: `-edit` query command

Optional with: `-new` and `-find` query commands

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-op=eq&FirstName=Sam&-max=1&-find
```

Note Specifying a field name multiple times in a single request is not supported; FileMaker Server parses all of the values, but uses only the last value parsed.

fieldname.op (Comparison operator) query parameter

Specifies the comparison operator to apply to the field name that precedes the operator. Comparison operators are used with the `-find` query command.

Value is: The operator you want to use. Valid operators are as follows:

Keyword	FileMaker Pro equivalent operator
`eq`	**=word**
`cn`	***word***
`bw`	**word***
`ew`	***word**
`gt`	**> word**
`gte`	**>= word**
`lt`	**< word**
`lte`	**<= word**
`neq`	**omit, word**

Optional with: `-find` query command

Requires: A field name and a value

The syntax for specifying a comparison operator is:

`table-name::field-name=value&table-name::field-name.op=op-symbol`

where:

- `table-name` is the table that contains the field and is only required if the field is not in the source table of the layout specified in the query string.
- `op-symbol` is one of the keywords in the preceding table, such as `cn`.

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&name=Tim&name.op=cn&-find
```

Note The `bw` keyword does not work with date, time, or timestamp strings, or with the current date (`//`) find operator.

You can use any FileMaker Pro find operator by including it as part of the search criteria instead of specifying the fieldname.op operator keyword. For example, to find a range of values using the range (. . .) find operator, do not specify any operator keyword. Instead, use the characters ". . ." between the range values in the search criteria.

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&IDnum=915...925&-find
```

For information on the operators you can use to find text, see FileMaker Pro Help.

-lay (Layout) query parameter

Specifies the database layout you want to use.

Value is: Name of the layout

Required with: All query commands except `-dbnames`, `-layoutnames`, and `-scriptnames`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-view
```

-lay.response (Switch layout for response) query parameter

Specifies that FileMaker Server should use the layout specified by the `-lay` parameter when processing a request, and switch to the layout specified by the `-lay.response` parameter when processing the XML response.

If you don't include the `-lay.response` parameter, FileMaker Server uses the layout specified by the `-lay` parameter when processing both the request and the response.

You can use the `-lay.response` parameter for XML requests.

Value is: Name of the layout

Optional with: All query commands except `-dbnames`, `-layoutnames`, and `-scriptnames`.

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=Budget&Salary=100000&Salary.op=gt&-find&-lay.response=ExecList
```

-lop (Logical operator) query parameter

Specifies how the find criteria in the `-find` query command are combined as either an “and” or an “or” search.

Value is: `and` or `or`
If the `-lop` query parameter is not included, then the `-find` query command uses the “and” value.

Optional with: `-find` query command

Note Not supported by `-findquery` query command.

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&Last+Name=Smith&Birthdate=2/5/1972&-lop=and&-find
```

-max (Maximum records) query parameter

Specifies the maximum number of records you want returned.

Value is: A number, or use the value `all` to return all records. If `-max` is not specified, all records are returned.

Optional with: `-find`, `-findall`, and `-findquery` query commands

Note The `-max` query parameter does not affect the values returned for portal records. To limit the number of rows returned for portal records, see “–relatedsets.max (Limit portal records) query parameter” on page 59.

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-max=10&-findall
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-max=all&-findall
```

-modid (Modification ID) query parameter

The modification ID is an incremental counter that specifies the current version of a record. By specifying a modification ID when you use an `-edit` query command, you can make sure that you are editing the current version of a record. If the modification ID value you specify does not match the current modification ID value in the database, the `-edit` query command is not allowed and an error code is returned.

Value is: A modification ID, which is a unique identifier for the current version of a record in a FileMaker database.

Optional with: `-edit` query command

Requires: `-recid` parameter

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-recid=22&-modid=6&last_name=Jones&-edit
```

-query (Compound find request) query parameter

Specifies the query names and search criteria for a compound find request. See “–findquery (Compound find) query command” on page 49.

Value is: A query expression.

Required with: `-findquery` query command

The syntax for a compound find request is:

`-query=<request-declarations><request-definitions>&-findquery`

where:

`<request-declarations>` is two or more request declarations.

- Each request declaration is composed of one or more query identifiers separated by commas, and enclosed in parentheses. A query identifier is the letter “q” followed by a number. For example: `q1`
- Enclosed in parentheses, the multiple queries act as logical AND searches that narrow the found set. For example, `(q1, q2)` returns records that match `q1` and `q2`.

  Note It is not recommended to use the same fields for multiple `q` variables in the same “and” search criteria.

- As with FileMaker Pro, each request can be either a find request or an omit request. A find request adds the matching records to the found set; an omit request removes the matching records from the found set. The default is a find request. For an omit request, put an exclamation point (!) in front of the opening parenthesis.

  ### Example

  ```
  (q1);!(q2)
  ```

  `q1` is a find request; `q2` is an omit request because it is preceded by an exclamation point.

- Requests are separated by semicolons. Multiple find requests act as logical OR searches that broaden the found set. For example, `(q1);(q2)` returns records that match `q1` or `q2`. Omit requests do not act as logical OR searches because omit requests remove records from the found set.
- Requests are executed in the order specified; the found set includes the results of the entire compound find request.

`<request-definitions>` is a request definition for each request declaration. Each request definition consists of a search field and value definition. A minus (-) sign starts the request definition.

Syntax:

```
-<query-id>=<fieldname>&-<query-id>.value=<value>
```

Examples

```
-q1=typeofanimal&-q1.value=Cat
-q2=name&-q2.value=Fluffy
```

Find records of gray cats that are not named “Fluffy”:

```
http://host/fmi/xml/fmresultset.xml?-db=petclinic&-lay=Patients
&-query=(q1, q2);!(q3)&-q1=typeofanimal&-q1.value=Cat&-q2=color
&-q2.value=Gray&-q3=name&-q3.value=Fluffy&-findquery
```

-recid (Record ID) query parameter

Specifies the record you want processed. Used mainly by the `-edit`, and `-delete` query commands. Used by the `-view` command to retrieve related value list data in the FMPXMLLAYOUT grammar.

Value is: A record ID, which is a unique specifier to a record in a FileMaker database

Required with: `-edit`, `-delete`, and `-dup` query commands

Optional with: `-find` query and `-view` commands

Examples

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-recid=22&-delete
http://localhost/fmi/xml/FMPXMLLAYOUT.xml?-db=test&-lay=empty&-view
&-recid=9
```

–relatedsets.filter (Filter portal records) query parameter

Specifies whether to limit the portal records to be returned in the results for this query.

Value is: `layout` or `none`

- If `–relatedsets.filter` is set to `layout`, then the **Initial row** setting specified in the FileMaker Pro Portal Setup dialog box is respected.
  - If the **Allow vertical scrolling** setting is enabled in the Portal Setup dialog box, then use the `–relatedsets.max` option to specify the maximum number of records to be returned. See "–relatedsets.max (Limit portal records) query parameter" below.
  - If the **Allow vertical scrolling** setting is disabled or the `–relatedsets.max` option is not used, then the **Number of rows** setting in the Portal Setup dialog box determines the number of portal records to be returned.
- The default value is `none` if this parameter is not specified. If `–relatedsets.filter` is set to `none`, then the Web Publishing Engine returns all records in the portal. The values for **Initial row** and **Number of rows** specified in the Portal Setup dialog box are ignored.

Notes

- The `–relatedsets.filter` parameter has no impact on how portal records are sorted in XML queries. The sort specified in FileMaker Pro is respected whether the `–relatedsets.filter` parameter value is `layout` or `none`.
- The **Filter portal records** setting in the Portal Setup dialog box is not supported for XML queries. Any calculation specified for the **Filter portal records** setting is ignored.

Optional with: `–find`, `–edit`, `–new`, `–dup`, and `–findquery`.

Examples

```
http://localhost/fmi/xml/fmresultset.xml?-db=FMPHP_Sample&-lay=English
&-relatedsets.filter=none&-findany
http://localhost/fmi/xml/fmresultset.xml?-db=FMPHP_Sample
&-lay=English&relatedsets.filter=layout&-relatedsets.max=all&-findany
http://localhost/fmi/xml/fmresultset.xml?-db=FMPHP_Sample&-lay=English
&-relatedsets.filter=layout&-relatedsets.max=10&-findany
```

-relatedsets.max (Limit portal records) query parameter

Specifies the maximum number of portal records to return in the results for this query.

Value is: an integer, or `all`.

- The `-relatedsets.max` parameter is respected only if the **Allow vertical scrolling** setting is enabled in the FileMaker Pro Portal Setup dialog box and the `-relatedsets.filter` parameter is `layout`.
  - If the `-relatedsets.max` parameter specifies an integer, then the Web Publishing Engine returns that number of portal records starting with the initial row.
  - If the `-relatedsets.max` parameter specifies `all`, then the Web Publishing Engine returns all portal records.

Note For information on filtering portal records, see "–relatedsets.filter (Filter portal records) query parameter" above.

Optional with: `-find`, `-edit`, `-new`, `-dup`, and `-findquery`.

Examples

```
http://localhost/fmi/xml/fmresultset.xml?-db=FMPHP_Sample
&-lay=English&relatedsets.filter=layout&-relatedsets.max=all&-findany
http://localhost/fmi/xml/fmresultset.xml?-db=FMPHP_Sample&-lay=English
&-relatedsets.filter=layout&-relatedsets.max=10&-findany
```

-script (Script) query parameter

Specifies the FileMaker script to run after the query command and sorting are executed. See "Understanding how an XML request is processed" on page 41.

Value is: Script name

Optional with: All query commands except `-dbnames`, `-layoutnames`, and `-scriptnames`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-script=myscript&-findall
```

-script.param (Pass parameter to Script) query parameter

Passes a parameter to the FileMaker script specified by `-script`.

Value is: A single text parameter.

- To pass in multiple parameters, you can create a string delimiting the parameters and have your script parse out the individual parameters. For example, pass "`param1|param2|param3`" as a list with the "|" character URL-encoded as this: `param1%7Cparam2%7Cparam3`
- To treat the text parameter as a value that is not text, your script can convert the text value. For example, to convert the text value to a number, your script could include the following: `GetAsNumber(Get(ScriptParam))`
- If your query contains `-script.param` without `-script`, then `-script.param` is ignored.
- If your query contains more than one `-script.param`, then the Web Publishing Engine uses the last value that it parses.

Optional with: `-script`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-script=myscript&-script.param=Smith%7CChatterjee%7CSu
&-findall
```

-script.prefind (Script before Find) query parameter

Specifies the FileMaker script to run before finding and sorting of records (if specified) during processing of the `-find` query command.

Value is: Script name

Optional with: All query commands except `-dbnames`, `-layoutnames`, and `-scriptnames`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-script.prefind=myscript&-findall
```

-script.prefind.param (Pass parameter to Script before Find) query parameter

Passes a parameter to the FileMaker script specified by `-script.prefind`.

Value is: A single text parameter.

- To pass in multiple parameters, you can create a string delimiting the parameters and have your script parse out the individual parameters. For example, pass "`param1|param2|param3`" as a list with the "|" character URL-encoded as this: `param1%7Cparam2%7Cparam3`
- To treat the text parameter as a value that is not text, your script can convert the text value. For example, to convert the text value to a number, your script could include the following: `GetAsNumber(Get(ScriptParam))`
- If your query contains `-script.prefind.param` without `-script.prefind`, then `-script.prefind.param` is ignored.

- If your query contains more than one `-script.prefind.param`, then the Web Publishing Engine uses the last value that it parses.

Optional with: `-script.prefind`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-script.prefind=myscript&-script.prefind.param=payroll
&-findall
```

-script.presort (Script before Sort) query parameter

Specifies the FileMaker script to run after finding records (if specified) and before sorting records during processing of the `-find` query command.

Optional with: All query commands except `-dbnames`, `-layoutnames`, and `-scriptnames`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-script.presort=myscript&-sortfield.1=dept
&-sortfield.2=rating&-findall
```

-script.presort.param (Pass parameter to Script before Sort) query parameter

Passes a parameter to the FileMaker script specified by `-script.presort`.

Value is: A single text parameter.

- To pass in multiple parameters, you can create a string delimiting the parameters and have your script parse out the individual parameters. For example, pass "`param1|param2|param3`" as a list with the "|" character URL-encoded as this: `param1%7Cparam2%7Cparam3`
- To treat the text parameter as a value that is not text, your script can convert the text value. For example, to convert the text value to a number, your script could include the following: `GetAsNumber(Get(ScriptParam))`
- If your query contains `-script.presort.param` without `-script.presort`, then `-script.presort.param` is ignored.
- If your query contains more than one `-script.presort.param`, then the Web Publishing Engine uses the last value that it parses.

Optional with: `-script.presort`

Example

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-script.presort=myscript&-script.presort.param=18%7C65
&-sortfield.1=dept&-sortfield.2=rating&-findall
```

-skip (Skip records) query parameter

Specifies how many records to skip in the found set.

Value is: A number. If the value is greater than the number of records in the found set, then no record is displayed. The default value is 0.

Optional with: `-find` query command

Example

The first 10 records in the found set are skipped and records 11 through 15 are returned:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=departments&-skip=10&-max=5&-findall
```

-sortfield (Sort field) query parameter

Specifies the field to use for sorting.

Value is: field name

Optional with: `-find` or `-findall` query commands

The `-sortfield` query parameter can been used multiple times to perform multiple field sorts. The syntax for specifying the precedence of the sort fields is:

```
-sortfield.precedence-number=fully-qualified-field-name
```

where the `precedence-number` in the `-sortfield.precedence-number` query parameter is a number that specifies the precedence to use for multiple sort fields. The value for `precedence-number`:

- must start from 1
- must increment sequentially
- must not be greater than 9

Example

The “dept” field is sorted first, and then the “rating” field is sorted. Both fields are sorted in ascending order because the `-sortorder` query parameter is not specified.

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=performance&-sortfield.1=dept&-sortfield.2=rating&-findall
```

-sortorder (Sort order) query parameter

Indicates the direction of a sort.

Value is: The sort order. Valid sort orders are as follows, where `<value-list-name>` is a value list name such as `Custom`:

Keyword	FileMaker Pro Equivalent Operator
`ascend`	Sort a to z, -10 to 10
`descend`	Sort z to a, 10 to -10
`<value-list-name>`	Sort using the specified value list associated with the field on the layout

Optional with: `-find` or `-findall` query commands

Requires: `-sortfield` query parameter

The `-sortorder` query parameter can been used with the `-sortfield` query parameter to specify the sort order of multiple sort fields. The syntax for specifying the sort order of a sort field is:

```
-sortorder.precedence-number=sort-method
```

where:

- `precedence-number` in the `-sortorder.precedence-number` parameter is a number from 1 to 9 that specifies the `-sortfield` query parameter that the `-sortorder` query parameter applies to
- `sort-method` is one of the keywords in the preceding table to specify the sort order, such as ascend

Example

The sort order of the highest precedence sort field (dept) is ascend, and the sort order of the second highest precedence sort field (rating) is descend. The `precedence-number` 2 in `-sortorder.2` specifies that the query parameter `-sortorder.2=descend` applies to the `-sortfield.2=rating` query parameter.

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees
&-lay=performance&-sortfield.1=dept&-sortorder.1=ascend
&-sortfield.2=rating&-sortorder.2=descend&-findall
```

Note If a `-sortorder` query parameter is not specified for a sort field, the default ascending sort is used.

Chapter 6
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 webpage 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, or use your own version of PHP. (If you decide to use your own version of PHP, see "Manually installing the FileMaker API for PHP" on page 66.)
- 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, multistep scripts. FileMaker supports many 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 (macOS)—the FileMaker Web Server Module is installed on the web server
  - the FileMaker Web Publishing Engine
  - the FileMaker Database Server
- 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
- PHP installed on the web server—FileMaker Server can install the supported version of PHP, or you can use your own version
  - For the minimum required version of PHP, see the FileMaker Server technical specifications.
  - For information about PHP, see php.net.
  - The version of PHP installed on the web server must support cURL (client URL library) functions. For information about cURL, see php.net/curl.

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

See FileMaker Server Installation and Configuration 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. 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.

Note For the minimum required version of PHP, see the FileMaker Server technical specifications. 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 (macOS):
  `/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 folder where your PHP scripts reside.
  - For IIS (Windows) through HTTP or HTTPS:
    `[drive]:\Program Files\FileMaker\FileMaker Server\HTTPServer\Conf`
    where `[drive]` is the drive on which the Web Publishing Engine component of your FileMaker Server deployment resides.
  - For Apache (macOS) through HTTP:
    `/Library/FileMaker Server/HTTPServer/htdocs`
- For Apache (macOS) through HTTPS:
  `/Library/FileMaker Server/HTTPServer/htdocs/httpsRoot`

  Note If you enable **Use HSTS for web clients** in Admin Console, use the HTTPS directory for hosting PHP site files.

- one of the include_path directories in your PHP installation. The default location for macOS is `/usr/lib/php`.

Where to go from here

- Use FileMaker Server Admin Console to enable Custom Web Publishing. See FileMaker Server Help and FileMaker Server Installation and Configuration Guide.
- In FileMaker Pro, open each FileMaker database that you want to publish, and make sure the database has the appropriate extended privileges enabled for Custom Web Publishing. See “Enabling Custom Web Publishing in a database” on page 13.
- To learn how to access data in FileMaker databases using the FileMaker API for PHP, see chapter 8, “Using the FileMaker API for PHP.”

Chapter 7
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 8, “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. See FileMaker Server Installation and Configuration Guide. To support PHP solutions, a PHP engine is installed with the web server on the master machine. FileMaker Server hosts a PHP solution when you place the PHP files on the master machine’s web server.

- 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

1. In Admin Console, make sure **Enable PHP publishing** is selected. See FileMaker Server Installation and Configuration Guide.

2. In 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 2, “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 8, “Using the FileMaker API for PHP.”

4. Copy or move your site directory structure and files to the following folder on the web server of the master machine.
   - For IIS (Windows) through HTTP or HTTPS:
     `[drive]:\Program Files\FileMaker\FileMaker Server\HTTPServer\Conf`
     where `[drive]` is the drive on the master machine where your FileMaker Server deployment resides.
   - For Apache (macOS) through HTTP:
     `/Library/FileMaker Server/HTTPServer/htdocs`
   - For Apache (macOS) through HTTPS:
     `/Library/FileMaker Server/HTTPServer/htdocs/httpsRoot`

   Note Use the HTTPS directory to host PHP site files when **Use HSTS for web clients** is enabled in Admin Console. With **Use HSTS for web clients** enabled, once a web client connects to FileMaker Server using HTTPS, the web browser prevents the client from using an HTTP connection for all web content hosted by FileMaker Server.

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 “About 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.

   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 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 9, “Staging, testing, and monitoring a site.”

Chapter 8
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 66.

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 (macOS): `/Library/FileMaker Server/Documentation/PHP API Documentation/index.html`

FileMaker API for PHP support

You can find additional information about the FileMaker API for PHP on the FileMaker Support page.

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

Important Commands have different return values, as defined by the `FileMaker.php` class. For example, some commands return the Boolean value TRUE or a FileMaker_Error object. Other commands return a FileMaker_Result object that could contain an entire "found set" of records in a layout. To avoid computer memory overload problems, be aware of the expected return values of the commands you use. See the "FileMaker API for PHP Reference" for detailed information about return values for each command.

The basic tasks that most PHP applications need to perform are described in the following:

- “Working with records” on page 73
- “Running FileMaker scripts” on page 75
- “Performing find requests” on page 81

Decoding data for use in the FileMaker API

If your PHP application retrieves data from a website, that data may be URL-encoded. The FileMaker API for PHP expects data to be decoded strings, not URL-encoded strings. As a general practice, you may want to call the urldecode() function when retrieving data in your PHP application.

Example

```
$user = urldecode($_GET['user']);
$event = urldecode($_GET['event']);
```

Note Avoid using strings that include ampersand (&) characters with the FileMaker API for PHP. Use a backslash as an escape character before special characters in strings passed to the FileMaker API for PHP.

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.

Examples

Connecting to a server to get a list of databases :

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

Connecting to a specific database on a server:

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

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

Note The `hostspec` property defaults to the value `http://localhost`. Because the PHP engine is installed with the master machine’s web server component, there is no need to specify the hostspec property.

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.

  Example

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

  Using the FileMaker_Record `commit()` method, the `$result` variable is assigned the Boolean value TRUE and the new record is created in the FileMaker database when there are no errors.

  If an error occurs, the variable `$result` contains a FileMaker_Error object. Check for errors after executing the `commit()` method.

- 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.

  Example

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

  Using the FileMaker_Command `execute()` method, the `$result` variable contains a FileMaker_Result object with all the information about the created record when there are no errors.

  If an error occurs, the variable `$result` contains a FileMaker_Error object. Check for errors after executing the `execute()` method.

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.

  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.

  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.

  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.

  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 72.)

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.

Important When running a FileMaker script, the size of the FileMaker_Result object that is returned depends on behavior of the FileMaker script. For example, if a FileMaker script switches to a specified layout, then all of the records from that layout’s table may be in the found set, and all of the records in that found set could be returned in the FileMaker_Result object. To avoid computer memory overload problems, be aware of the data returned by a FileMaker script before running that FileMaker script in a PHP application.

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. See “Using the Find command” on page 82.

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. See “Using the Find command” on page 82.

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 nonexistent field, the request deletes the current record and returns error code 102 (“Field is missing”), but does not execute the script.

The layout specified for the `newFindCommand()` method is used when processing the find request. When the `setResultLayout()` method switches to another layout, the error object for the find request based on the original layout is no longer available. To test the error object from the find request based on the original layout, check the error object before changing the layout.

Example

```
request = $fm->newFindCommand('Students');
$request->addFindCriterion('Day', 'Wednesday');

// Perform the Find
$result = $request->execute();

if (FileMaker::isError($result)) {
  if ($result->code = 401) {
    $findError = 'There are no Records that match that request: '. ' (' .
     $result->code . ')';
  } else {
    $findError = 'Find Error: '. $result->getMessage(). ' (' . $result->code
     . ')';
  }
}
$request->setResultLayout('Teachers');
// Switch to the result layout
$result = $request->execute();
```

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 82.
- Find Any command. See “Using the Find Any command” on page 82.
- Find command. See “Using the Find command” on page 82.
- Compound Find command. See “Using a Compound Find command” on page 83.

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.

Note Make sure the layout name is unique. If your database has two layouts with the same name, the FileMaker API for PHP cannot distinguish between them. In addition, the API is not case sensitive. For example, if your database has one layout named Websites and another layout named WebSites, the API cannot distinguish between them.

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

Examples

Finding a record by field name:

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

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. There are several ways 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 `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');

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

// 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')) .'">';
```

Limiting the portal rows returned by find requests

In a solution that has many related records, querying and sorting portal records can be time consuming. To limit 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 box are respected. The records are sorted based on the sort defined in the Portal Setup dialog box, with the record set filtered to start with the initial row specified.
- the minimum number of portal records returned: an integer value or `all`
  - This value is used only if the **Allow vertical scrolling** setting is enabled in the Portal Setup dialog box. 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 **Allow vertical scrolling** setting is disabled, the Portal Setup dialog box's **Number of rows** setting determines the maximum number of related records that are returned.

Note The **Filter portal records** setting in the Portal Setup dialog box is not supported for PHP queries. Any calculation specified for the **Filter portal records** setting is ignored.

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, `4/31/2017` 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\30-2017` 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";
  }
}
```

Result

```
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.”

Chapter 9
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, you must copy or move the required files to the correct locations on the staging server(s).

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

   - IIS (Windows) through HTTP or HTTPS:
     `[drive]:\Program Files\FileMaker\FileMaker Server\HTTPServer\Conf`
     where `[drive]` is the drive of your FileMaker Server master machine.
   - Apache (macOS) through HTTP:
     `/Library/FileMaker Server/HTTPServer/htdocs`
   - Apache (macOS) through HTTPS:
     `/Library/FileMaker Server/HTTPServer/htdocs/httpsRoot`

   Note If you enable **Use HSTS for web clients** in Admin Console, use the HTTPS directory for hosting site files.

5. If you have not already done so, copy or move any referenced container field objects to the appropriate directory on the master 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.
   - When you use FileMaker Pro to upload a database with container fields that store objects externally, the externally stored container field data is uploaded to FileMaker Server as part of the process. See FileMaker Pro Help for information on transferring the database files to FileMaker Server.
   - When 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.
6. Copy any additional components of your web application to the master machine. For Custom Web Publishing with XML, your web application processes the XML data before sending it to another application or to the client.

Testing a Custom Web Publishing site

Before notifying users that your Custom Web Publishing site is available, verify that it looks and functions as you expect.

- 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 You can view and test your site on the master machine without using a network connection by using http://127.0.0.1/ in the URL.

- For PHP solutions, use http://127.0.0.1/<site_path>
  where <site_path> is the relative path to the homepage for your site.
- For information on the URL syntax in XML solutions, see "About the URL syntax for XML data and container objects" on page 25.

Stylesheets for testing XML output

Examples

Here are two examples of XSLT stylesheets that are useful for testing XML output.

The following stylesheet example outputs the requested XML data without doing any transformation. This stylesheet is useful for displaying the actual XML data that the Web Publishing Engine is using.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fmrs="http://www.filemaker.com/xml/fmresultset">
  <xsl:output method="xml"/>
  <xsl:template match="/">
    <xsl:copy-of select="."/>
  </xsl:template>
</xsl:stylesheet>
```

When debugging a stylesheet, you can use the following example of an HTML <textarea> tag to display the XML source document that was accessed via the stylesheet in a scrolling text area.

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="1.0"
 xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
 xmlns:fmrs="http://www.filemaker.com/xml/fmresultset">
  <xsl:output method="html"/>
  <html>
    <body>
      <xsl:template match="/fmrs:fmresultset">
        <textarea rows="20" cols="100">
          <xsl:copy-of select="."/>
        </textarea><br/>
      </xsl:template>
    </body>
  </html>
</xsl:stylesheet>
```

Monitoring your site

You can 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. See the documentation for the Microsoft IIS web server.

Apache (macOS): 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. 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 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.

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

- Windows:
  `[drive]:\Program Files\FileMaker\FileMaker Server\Logs\wpe.log`
  where `[drive]` is the primary drive from which the system is started.
- macOS: `/Library/FileMaker Server/Logs/wpe.log`

Web Publishing Engine log settings

The wpe.log file is generated if the **Enable logging for Web Publishing** setting is enabled in 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 Admin Console, see FileMaker Server Help.

Important Over time, the wpe.log file may become very large. Use 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. 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 master machine.
- `[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 99).

  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:

```
2017-06-02 15:15:31 -0700   -   -   -   -   INFO   -   -   FileMaker Server
Web Publishing Engine started.
2017-06-02 15:46:52 -0700   -   -   -   -   INFO   -   -   FileMaker Server
Web Publishing Engine stopped.
```

Successful or failed XML query requests:

```
2017-06-02 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"
2017-06-02 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:

```
2017-06-02 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:

```
2017-06-09 10:59:49 -0700  WPC_SERVER  192.168.100.101:0  jdoe  -  INFO
-   -   XML Web Publishing Engine is enabled.
```

System errors:

```
2017-06-02 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 file is called web_server_module_log.txt and is located in the Logs folder in the FileMaker Server folder on the web server host.

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:

- Windows:
  - `[drive]:\Program Files\FileMaker\FileMaker Server\Admin\admin-master-tomcat\logs\`
    where `[drive]` is the primary drive from which the system is started.
  - `[drive]:\Program Files\FileMaker\FileMaker Server\Web Publishing\publishing-engine\jwpc-tomcat\logs`
    where `[drive]` is the primary drive from which the system is started.
- macOS:
  - `/Library/FileMaker Server/Admin/admin-master-tomcat/logs/`
  - /Library/FileMaker Server/Web Publishing/publishing-engine/jwpc-tomcat/logs

Appendix A
Error codes for Custom Web Publishing

The Web Publishing Engine generates error codes for database and query string errors that may occur during an XML data request.

For a list of updated error codes, see the FileMaker Knowledge Base.

Error code numbers in XML format

The Web Publishing Engine generates an error code for databases published in XML format whenever data is requested. This type of error code value is inserted at the beginning of the XML document in the `<error code>` element for the `fmresultset` grammar, or in the `<ERRORCODE>` element for the `FMPXMLRESULT` or `FMPXMLLAYOUT` grammars. An error code of 0 indicates that no error has occurred.

Examples

Database error code in the `fmresultset` grammar:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE fmresultset PUBLIC "-//FMI//DTD fmresultset//EN"
 "http://192.168.123.101/fmi/xml/fmresultset.dtd">
<fmresultset xmlns="http://www.filemaker.com/xml/fmresultset"
version="1.0">
  <error code="0"></error>
```

Database error code in the `FMPXMLRESULT` grammar:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FMPXMLRESULT PUBLIC "-//FMI//DTD FMPXMLRESULT//EN"
 "http://192.168.123.101/fmi/xml/FMPXMLRESULT.dtd">
<fmpxmlresult xmlns="http://www.filemaker.com/fmpxmlresult">
  <ERRORCODE>0</ERRORCODE>
```

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

Error code numbers for FileMaker databases

For FileMaker Pro error codes, see FileMaker Pro Help.

FileMaker Server provides error code 959 to indicate that a technology has been disabled.

For example, if the server administrator disables Custom Web Publishing with XML using the **Web Publishing** > **XML** tab in FileMaker Server Admin Console, then XML queries return the error code 959.

Example

Error code 959 in the `FMPXMLLAYOUT` grammar:

```
<FMPXMLLAYOUT>
  <ERRORCODE>959</ERRORCODE>
  <LAYOUT DATABASE="" NAME=""/>
  <VALUELISTS/>
</FMPXMLLAYOUT>
```

Index

F

G

H

I

K

L

M

N

O

P

Q

R

S

T

U

V

W

X