

FileMaker® Server 11

Custom Web Publishing
with XML and XSLT



© 2007–2010 FileMaker, Inc. All Rights Reserved.

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

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

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

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

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

Edition: 01

Contents

About this guide	9
Chapter 1	
<i>Introducing Custom Web Publishing</i>	
About the Web Publishing Engine	12
How a Web Publishing Engine request is processed	12
Custom Web Publishing with PHP	13
Custom Web Publishing with XML and XSLT	13
Comparing PHP to XML and XSLT	14
Reasons to choose PHP	14
Reasons to choose XML and XSLT	14
Chapter 2	
<i>About Custom Web Publishing with XML and XSLT</i>	
Creating dynamic websites with the Web Publishing Engine	15
About Custom Web Publishing with XML	15
About Custom Web Publishing with XSLT	16
About developing XSLT stylesheets	16
Key features in Custom Web Publishing with XML and XSLT	16
Web publishing requirements	17
What is required to publish a database using Custom Web Publishing	17
What web users need to access a Custom Web Publishing solution	17
Connecting to the Internet or an intranet	17
Where to go from here	18
Chapter 3	
<i>Preparing databases for Custom Web Publishing</i>	
Enabling Custom Web Publishing in a database	19
Accessing a protected database	19
Protecting your published databases	20
Web server support for Internet media types (MIME)	20
About publishing the contents of container fields on the web	21
Publishing container field objects stored in a database	21
Publishing container field objects stored as a file reference	21
How web users view container field data	21

FileMaker scripts and Custom Web Publishing	22
Script tips and considerations	22
Script behavior in Custom Web Publishing solutions	23
Script triggers and Custom Web Publishing solutions	23

Chapter 4

Introduction to Custom Web Publishing with XSLT

About FileMaker XSLT stylesheets	25
What are some examples of using FileMaker XSLT stylesheets?	25
Getting started using Custom Web Publishing with XSLT	26
How the Web Publishing Engine generates pages based on XML data and XSLT stylesheets	26
General steps for using Custom Web Publishing with XSLT	27
Using FileMaker XSLT Site Assistant to generate FileMaker XSLT stylesheets	28
Before using XSLT Site Assistant	28
Starting XSLT Site Assistant	29
Using XSLT Site Assistant	29
About XSLT Site Assistant's generated stylesheets	29
Using FileMaker XSLT stylesheets in a website or program	30
Troubleshooting XSLT stylesheets	31

Chapter 5

Accessing XML data with the Web Publishing Engine

Using Custom Web Publishing with XML	33
Differences between the Web Publishing Engine and FileMaker Pro XML Import/Export	33
How the Web Publishing Engine generates XML data from a request	34
General process for accessing XML data from the Web Publishing Engine	35
About the URL syntax for XML data and container objects	35
About the URL syntax for XML data	35
About the URL syntax for FileMaker container objects in XML solutions	36
About URL text encoding	37
Accessing XML data via the Web Publishing Engine	37
About namespaces for FileMaker XML	38
About FileMaker database error codes	38
Retrieving the document type definitions for the FileMaker grammars	38
Using the fmsresultset grammar	39
Description of elements in the fmresultset grammar	39
Example of XML data in the fmresultset grammar	41

Using other FileMaker XML grammars	42
Description of elements in the FMPXMLRESULT grammar	42
Example of XML data in the FMPXMLRESULT grammar	43
Description of elements in the FMPXMLLAYOUT grammar	43
Example of XML data in the FMPXMLLAYOUT grammar	45
About UTF-8 encoded data	46
Using FileMaker query strings to request XML data	46
Switching layouts for an XML response	48
Understanding how an XML request is processed	48
Using server-side and client-side processing of stylesheets	49
Troubleshooting XML document access	50

Chapter 6

Developing FileMaker XSLT stylesheets

Using XSLT stylesheets with the Web Publishing Engine	51
About the FileMaker XSLT Extension Function Reference	52
About the FileMaker XSLT Starter Solution	52
About the URL syntax for FileMaker XSLT stylesheets	52
About the URL syntax for FileMaker container objects in XSLT solutions	53
Using query strings in FileMaker XSLT stylesheets	54
Specifying an XML grammar for a FileMaker XSLT stylesheet	54
About namespaces and prefixes for FileMaker XSLT stylesheets	55
Using statically defined query commands and query parameters	55
Setting text encoding for requests	57
Specifying an output method and encoding	58
About the encoding of XSLT stylesheets	58
Processing XSLT requests that do not query FileMaker Server	58
Using tokens to pass information between stylesheets	59

Using the FileMaker XSLT extension functions and parameters	59
About the FileMaker-specific XSLT parameters set by the Web Publishing Engine	59
Accessing the query information in a request	60
Obtaining client information	60
Using the Web Publishing Engine base URI parameter	61
Using the authenticated base URI parameter	61
Loading additional documents	61
Using the layout information for a database in a stylesheet	62
Using content buffering	63
Using Web Publishing Engine sessions to store information between requests	64
Using the session extension functions	64
Sending email messages from the Web Publishing Engine	66
Using the header functions	67
Using the cookie extension functions	68
Using the string manipulation extension functions	69
Comparing strings using Perl 5 regular expressions	70
Checking for values in a field formatted as a checkbox	70
Using the date, time, and day extension functions	72
Checking the error status of extension functions	76
Using logging	76
Using server-side processing of scripting languages	76
Defining an extension function	76
An extension function example	77

Chapter 7

Staging, testing, and monitoring a site

Staging a Custom Web Publishing site	81
Testing a Custom Web Publishing site	82
Examples of stylesheets for testing XML output	83
Monitoring your site	83
Using the web server access and error logs	84
Using the Web Publishing Engine application log	84
Using the Web Server Module error log	84
Using Web Publishing Core internal access logs	85

Appendix A

Valid names used in query strings

About the query commands and parameters	87
Guidelines for using query commands and parameters	88
About the FileMaker Query Strings Reference	88
About the syntax for a fully qualified field name	89
Using query commands with portal fields	89
About the syntax for specifying a global field	91
Query command reference	91
–dbnames (Database names) query command	91
–delete (Delete record) query command	91
–dup (Duplicate record) query command	92
–edit (Edit record) query command	92
–find, –findall, or –findany (Find records) query commands	92
–findquery (Compound find) query command	93
–layoutnames (Layout names) query command	93
–new (New record) query command	93
–process (Process XSLT stylesheets)	94
–scriptnames (Script names) query command	94
–view (View layout information) query command	94

Query parameter reference	95
–db (Database name) query parameter	95
–delete.related (Portal records delete) query parameter	95
–encoding (Encoding XSLT request) query parameter	95
–field (Container field name) query parameter	95
fieldname (Non-container field name) query parameter	96
fieldname.op (Comparison operator) query parameter	97
–grammar (Grammar for XSLT stylesheets) query parameter	98
–lay (Layout) query parameter	98
–lay.response (Switch layout for response) query parameter	98
–lop (Logical operator) query parameter	98
–max (Maximum records) query parameter	99
–modid (Modification ID) query parameter	99
–query (Compound find request) query parameter	99
–recid (Record ID) query parameter	100
–relatedsets.filter (Filter portal records) query parameter	100
–relatedsets.max (Limit portal records) query parameter	101
–script (Script) query parameter	101
–script.param (Pass parameter to Script) query parameter	101
–script.prefind (Script before Find) query parameter	102
–script.prefind.param (Pass parameter to Script before Find) query parameter	102
–script.presort (Script before Sort) query parameter	102
–script.presort.param (Pass parameter to Script before Sort) query parameter	103
–skip (Skip records) query parameter	103
–sortfield (Sort field) query parameter	104
–sortorder (Sort order) query parameter	104
–stylehref (Style href) query parameter	105
–styletype (Style type) query parameter	105
–token.[string] (Pass values between XSLT stylesheets) query parameter	105

Appendix B

Error codes for Custom Web Publishing

Error code numbers for FileMaker databases	107
Error code numbers for the Web Publishing Engine	114
Error code numbers for the FileMaker XSLT extension functions	115

Index

117

Preface

About this guide

This guide assumes you are experienced with XML and XSLT, developing websites, and using FileMaker® Pro to create databases. You should understand the basics of FileMaker Pro database design, and should understand the concepts of fields, relationships, layouts, portals, and containers. This guide provides the following information about Custom Web Publishing with XML and XSLT on FileMaker Server:

- what is required to develop a Custom Web Publishing solution using XML or XSLT
- how to publish your databases using XML or XSLT
- what web users need to access a Custom Web Publishing solution
- how to obtain XML data from databases hosted by FileMaker Server
- how to develop FileMaker XSLT stylesheets

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

The documentation for FileMaker Server includes the following information:

For information about	See
Installing and configuring FileMaker Server	<i>FileMaker Server Getting Started Guide</i> FileMaker Server Help
Instant Web Publishing	<i>FileMaker Instant Web Publishing Guide</i>
Custom Web Publishing with PHP	<i>FileMaker Server Custom Web Publishing with PHP</i>
Using PHP Site Assistant	PHP Site Assistant Help
Custom Web Publishing with XML and XSLT	<i>FileMaker Server Custom Web Publishing with XML and XSLT</i> (this book)
Using XSLT Site Assistant	XSLT Site Assistant Help
Installing and configuring ODBC and JDBC drivers, and using ODBJ and JDBC	<i>FileMaker ODBC and JDBC Guide</i>
How FileMaker Server Auto Update can download the most current plug-in to FileMaker Pro database client computers	<i>FileMaker Guide to Updating Plug-ins</i>

Chapter 1

Introducing Custom Web Publishing

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

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

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

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

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

With Custom Web Publishing, you can:

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

FileMaker Server provides two Custom Web Publishing technologies:

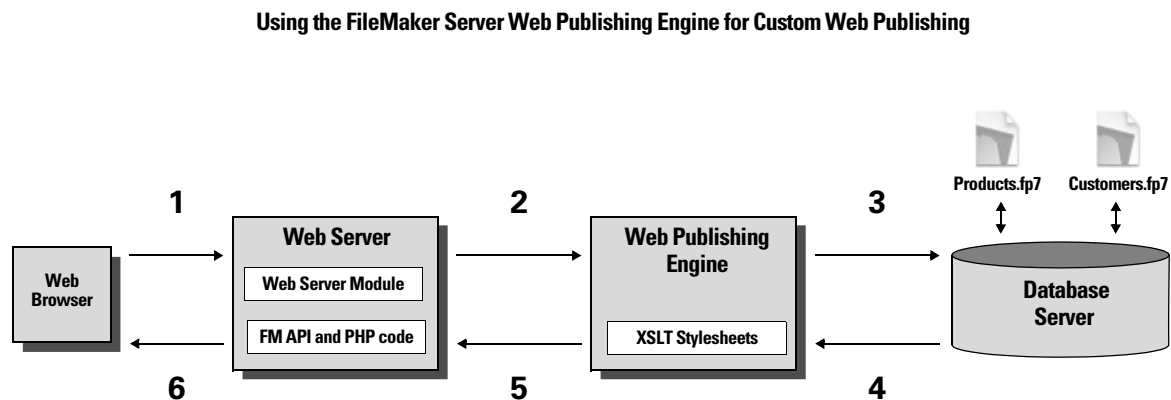
- **Custom Web Publishing with PHP:** Use the FileMaker API for PHP, which provides an object-oriented PHP interface to FileMaker Pro databases, to integrate your FileMaker data into a PHP web application. You can use PHP Site Assistant to generate a complete PHP website, or code PHP web pages yourself.
- **Custom Web Publishing with XML and XSLT:**
 - Use XML data publishing to exchange FileMaker data with other websites and applications.
 - Use server-processed XSLT stylesheets to integrate any subset of FileMaker data into other websites, with other middleware, and with custom applications. You can use XSLT Site Assistant to generate XSLT stylesheets, or code the stylesheets yourself.

About the Web Publishing Engine

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

Custom Web Publishing with XML and XSLT: The Web Publishing Engine functions as an XSLT processor, provides output as HTML, XML, or text (such as vCards) to the web server, which provides the output to the web browser. 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 URL can access XML data or reference an XSLT stylesheet. The Web Publishing Engine returns the XML data specified in the query string request, or the results of the referenced XSLT stylesheet.

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.



How a Web Publishing Engine request is processed

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

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

Custom Web Publishing with PHP

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

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

In addition, you can use FileMaker PHP Site Assistant to create PHP code that contains all of the necessary prerequisites and functions for properly accessing data in a FileMaker Pro database. PHP Site Assistant generates a multiple-page website that enables web users to search a database, view a list of records, browse records, sort records, add records, edit records, duplicate records, delete records, and view a summary report. FileMaker developers who have little PHP experience can use PHP Site Assistant to generate a complete PHP website. PHP developers who have little experience with FileMaker can use PHP Site Assistant to understand the FileMaker API for PHP objects and methods.

Custom Web Publishing with XML and XSLT

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

FileMaker Custom Web Publishing with XSLT lets you transform, filter, or format XML data for web browsers or other applications. You can:

- use an XSLT stylesheet to transform the data between a FileMaker XML grammar and another XML grammar in other applications or databases.
- filter the data by controlling which database fields are published by the stylesheet.
- format how the data is presented in a web page, and control how the web user interacts with the data.

The Web Publishing Engine uses your stylesheets to obtain data from a FileMaker database whenever a web user sends an HTTP request and a URL that references one of your XSLT stylesheets. The Web Publishing Engine uses a stylesheet to transform and format the XML data, and generates the resulting HTML page that the web user can work with.

In addition, you can use FileMaker XSLT Site Assistant to create basic XSLT stylesheets as a starting point for Custom Web Publishing with XSLT. XSLT Site Assistant generates stylesheets for pages that search the database, browse one record at a time, list the records in the database, sort records, add records, edit records, duplicate records, delete records, and display a summary report.

Comparing PHP to XML and XSLT

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

Reasons to choose PHP

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

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

Reasons to choose XML and XSLT

- FileMaker XML request parameter syntax is designed for database interaction, simplifying solution development.
- XML and XSLT are W3C standards.
- 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.
- XSLT lets you transform XML output into structured text documents such as RSS, RTF, vCard.
- You can use XSLT to transform XML output from one grammar to another.
- Templates make it easy to apply conditional formatting to variable data.
- You can use FMPXMLRESULT based stylesheets for Custom Web Publishing and for XML export from FileMaker Pro databases.
- FileMaker Server handles FileMaker XSLT stylesheet processing, preventing unauthorized access to data that might be unprotected using client-side XSLT stylesheets.

Chapter 2

About Custom Web Publishing with XML and XSLT

Creating dynamic websites with the Web Publishing Engine

The Web Publishing Engine provides Custom Web Publishing for FileMaker Server using XML data publishing and server-processed XSLT stylesheets. 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:** By using FileMaker XSLT stylesheets, 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 Instant Web Publishing, XML web publishing, or XSLT web publishing for all databases hosted by the server. As the FileMaker database owner, you can control web user access to Instant Web Publishing, XML web publishing, or XSLT web publishing for each database.
- **Server-side stylesheets:** Server-side XSLT stylesheet processing prevents unauthorized examination of confidential database information that might otherwise be possible with client-side stylesheets.
- **Control and filtering of published data:** By using XSLT stylesheets, 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 open standards:** You have more access to tools, resources and skilled personnel for Custom Web Publishing solutions. If you know standard XML or XSLT, then you can immediately 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.

About Custom Web Publishing with XML

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, or apply an XSLT stylesheet to the XML data. See chapter 5, “Accessing XML data with the Web Publishing Engine.”

About Custom Web Publishing with XSLT

Custom Web Publishing with XSLT provides the ability to transform, filter, or format XML data for use in a web browser or in other applications. You can use an XSLT stylesheet to transform the data between a FileMaker XML grammar and another XML grammar for use in another application or database. You can filter the data by controlling which database fields are published by the stylesheet. You can format how the data is presented in a web page, and control how the web user interacts with the data. See chapter 4, “Introduction to Custom Web Publishing with XSLT.”

The Web Publishing Engine uses your stylesheets to dynamically obtain data from a FileMaker database whenever a web user sends an HTTP request and a URL that references one of your XSLT stylesheets. The Web Publishing Engine uses a stylesheet to transform and format the XML data, and generates the resulting HTML page that the web user can work with.

For additional information about using FileMaker Server Custom Web Publishing with XML and XSLT, visit www.filemaker.com/documentation.

About developing XSLT stylesheets

FileMaker Server includes a tool for developing XSLT stylesheets. FileMaker XSLT Site Assistant is an application you can use to create basic XSLT stylesheets as a starting point for Custom Web Publishing with XSLT. XSLT Site Assistant is a good way to learn how FileMaker XSLT stylesheets are constructed. You can then use your own XSLT stylesheet authoring tools to modify the stylesheets as necessary. See “Using FileMaker XSLT Site Assistant to generate FileMaker XSLT stylesheets” on page 28.

Note FileMaker Server supports XSLT 1.0 as defined by the World Wide Web Consortium. Any XSLT authoring tools you use must produce standards-conforming XSLT 1.0.

Key features in Custom Web Publishing with XML and XSLT

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

- Databases are hosted on FileMaker Server, and FileMaker Pro is not required to be running.
- You can use server-side XSLT stylesheet processing, which is more secure than client-side stylesheet processing.
- You can use server-side processing of JavaScript in XSLT stylesheets. For information, see “Using server-side processing of scripting languages” on page 76.
- You can prevent the unauthorized use of query commands and query parameters with your FileMaker XSLT stylesheet by statically defining the query commands, parameters, and values that you want to use when XML data is requested. See “Using statically defined query commands and query parameters” on page 55.
- 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 20.
- Web users can perform complex, multi-step scripts. FileMaker supports about 70 script steps in Custom Web Publishing. See the section “FileMaker scripts and Custom Web Publishing” on page 22.
- You can pass a parameter value to a FileMaker script. For more information, see “`–script.param` (Pass parameter to Script) query parameter” on page 101, “`–script.prefind.param` (Pass parameter to Script before Find) query parameter” on page 102, and “`–script.presort.param` (Pass parameter to Script before Sort) query parameter” on page 103.

- The `fmresultset` XML grammar enables you to access fields by name and manipulate `relatedset` (portal) data.
- Using session functions in an XSLT stylesheet, you can store a web user's information and transactions in server-maintained sessions.
- To access data in a database, you must specify a layout. See appendix A, "Valid names used in query strings."
- Each web user can have a unique global field value that persists as long as a session is active. For general information on global fields, see FileMaker Pro Help. For information on using global fields with Custom Web Publishing, see "About the syntax for specifying a global field" on page 91.

Web publishing requirements

What is required to publish a database using Custom Web Publishing

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

- a FileMaker Server deployment that includes
 - a web server, either Microsoft IIS (Windows) or Apache (Mac OS X)
 - 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

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

What web users need to access a Custom Web Publishing solution

To access a Custom Web Publishing solution that uses XML or XSLT, 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 Getting Started Guide*.
- In FileMaker Pro, open each FileMaker database that you want to publish and make sure the database has the appropriate extended privilege(s) enabled for Custom Web Publishing. See “Enabling Custom Web Publishing in a database” on page 19.
- To learn how to access data in FileMaker databases using XML, see chapter 5, “Accessing XML data with the Web Publishing Engine.”
- To learn how to get started developing FileMaker XSLT stylesheets, see chapter 4, “Introduction to Custom Web Publishing with XSLT.”

Chapter 3

Preparing databases for Custom Web Publishing

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

Enabling Custom Web Publishing in a database

You must enable Custom Web Publishing in each database you want to publish. You can individually enable Custom Web Publishing with XML or Custom Web Publishing with XSLT, or you can enable both technologies in each database. If you don't enable one or both of these technologies 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 one or both of these extended privileges to one or more privilege sets:
 - To allow Custom Web Publishing with XML, use this keyword: `fmxml`
 - To allow Custom Web Publishing with XSLT, use this keyword: `fmxmlt`Since FileMaker Pro 8, the keywords `fmxml` and `fmxmlt` are defined on the Extended Privileges tab for you.
3. Assign the privilege set(s) that include the Custom Web Publishing extended privileges 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

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.

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 with XML or XSLT, you can limit who can access your published databases.

- Assign passwords to database accounts that are used for Custom Web Publishing.
- Enable Custom Web Publishing with XML or XSLT only in the privilege sets for accounts that you want to allow access to your published databases.
- To enable or disable a type of Custom Web Publishing technology for an individual database, set the extended privilege.
- Enable or disable a type of Custom Web Publishing technology for all Custom Web Publishing solutions in the Web Publishing Engine using FileMaker Server Admin Console. See 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.
- Use Secure Sockets Layer (SSL) encryption for communications between your web server and web users’ browsers. SSL encryption converts information exchanged between servers and clients into unintelligible information through using mathematical formulas known as ciphers. These ciphers are used to transform the information back into understandable data through encryption keys. For information on enabling and configuring SSL, see the documentation for your web server.

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

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. For more information, see the documentation for your web server.

About publishing the contents of container fields on the web

The contents of a container field, such as an image file, can be stored inside a FileMaker database, or stored as a file reference using a relative path.

Note The Web Publishing Engine does not support movie file streaming. Web users must download an entire movie file before being able to view the movie.

Publishing container field objects stored 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 36, and "About the URL syntax for FileMaker container objects in XSLT solutions" on page 53.

Publishing container field objects stored as a file reference

If a container field stores file references instead of actual files, follow these steps to publish the container field objects.

Note All QuickTime movies are stored in a container field as a reference.

To publish container field objects that are stored as a file reference:

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, move the files to: `<drive>:\inetpub\wwwroot`
 - For Apache, move the files to: `/Library/WebServer/Documents`

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. For more information, see the documentation for your web server.

How web users view container field data

When you publish a database on the web using the Web Publishing Engine, web users can work with data in container fields in these limited ways:

- Web users can't play sounds or display OLE objects in a container field—a graphic is displayed instead.
- Web users can't modify or add to the contents of container fields. Web users can't use a container field to upload data to the database.
- If your database contains graphics that aren't in GIF or JPEG format, the Web Publishing Engine creates a temporary JPEG image when the graphic data is requested by a web browser.

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 over 75 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 or in a `<?xslt-cwp-query?>` processing instruction in an XSLT stylesheet. To see script steps that are not supported, select **Web Publishing** from the **Show Compatibility** list in the Edit Script window in FileMaker Pro. Dimmed script steps are not supported on the web. 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 23. 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 application log file (`pe_application_log.txt`) for any scripting-related errors; for more information, see “Using the Web Publishing Engine application log” on page 84.

Keep these tips and considerations in mind:

- Use accounts and privileges to restrict the set of scripts that a web user can execute. Verify that the scripts contain only web-compatible script steps, and only provide access to scripts that should be used from a web browser.
- Consider the side effects of scripts that execute a combination of steps that are controlled by access privileges. For example, if a script includes a step to delete records, and a web user does not log in with an account that allows record deletion, the script will not execute the Delete Records script step. However, the script might continue to run, which could lead to unexpected results.
- In the Edit Script window, select **Run script with full access privileges** to allow scripts to perform tasks 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.
- If your scripts contain steps that are unsupported, for example, steps that are not web-compatible, use the **Allow User Abort** script step to determine how subsequent steps are handled.
 - If the **Allow User Abort** script step option is enabled (on), unsupported script steps stop the script from continuing.
 - If **Allow User Abort** is off, unsupported script steps are skipped and the script continues to execute.
 - If this script step is not included, scripts are executed as if the feature is enabled, so unsupported script steps stop scripts.
- Some scripts that work with one step from a FileMaker Pro client may require an additional **Commit Record/Request** step to save the data to the host. Because web users don't have a direct connection to the host, they aren't notified when data changes. For example, features like conditional value lists aren't as responsive for web users because the data must be saved to the host before the effects are seen in the value list field.

- Any script that modifies data should include the Commit Record/Request step, because data changes aren't visible in the browser until the data is saved or "submitted" to the server. This includes several script steps like Cut, Copy, Paste, and so on. Many single-step actions should be converted into scripts to include the Commit Record/Request step. When designing scripts that will be executed from a web browser, include the Commit Record/Request step at the end of a script to make sure all changes are saved.
- To create conditional scripts based on the type of client, use the Get(ApplicationVersion) function. If the value returned includes "Web Publishing Engine" then you know that the current user is accessing your database with Custom Web Publishing. For more information on functions, see FileMaker Pro Help.
- If you are using a script in an XSLT stylesheet that sets or modifies a state, you must use the FileMaker Server Admin Console to enable the XSLT Database Sessions option for the Web Publishing Engine. Otherwise, states are not maintained between requests. See FileMaker Server Help.

Script behavior in Custom Web Publishing solutions

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

Script step	Behavior in Custom Web Publishing solutions
Perform Script	Scripts cannot perform in other files, unless the files are hosted on FileMaker Server and Custom Web Publishing is enabled in the other files.
Exit Application	Logs off web users, closes windows, but does not exit the web browser application.
Allow User Abort	Determines how unsupported script steps are handled. Enable to stop scripts from continuing, and disable to skip unsupported steps. See "Script tips and considerations" on page 22 for more details. Web users cannot abort Custom Web Publishing scripts, but this option allows unsupported script steps to stop the script from continuing.
Set Error Capture	This is always enabled with Custom Web Publishing. Web users cannot abort Custom Web Publishing scripts.
Pause/Resume script	Although this script is supported in Custom Web Publishing, you should avoid using it. When a Pause step is executed, the script pauses. Only a script containing the Resume script step can make it resume execution. If the script remains in a paused state until the session times out, then the script will not be completed.
Sort Records	You must save a sort order with the Sort Records script step to execute in Custom Web Publishing.
Open URL	This script step has no effect in a Custom Web Publishing solution.
Go to Field	You cannot use Go to Field to make a particular field active in the web browser, but you can use this script step in conjunction with other script steps to perform tasks. For example, you can go to a field, copy the contents, go to another field and paste the value. To see the effect in the browser, be sure to save the record with the Commit Record script step.
Commit Record/Request	Submits the record to the database.

Script triggers and Custom Web Publishing solutions

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

Chapter 4

Introduction to Custom Web Publishing with XSLT

You can use FileMaker XSLT stylesheets to transform, filter, or format XML data for use in a web browser or in other programs and applications. This chapter introduces FileMaker XSLT stylesheets, and a tool to help you get started creating XSLT stylesheets—FileMaker XSLT Site Assistant. For more detailed information about how FileMaker XSLT stylesheets are constructed, see chapter 6, “Developing FileMaker XSLT stylesheets.”

About FileMaker XSLT stylesheets

You can use FileMaker XSLT stylesheets to:

- filter FileMaker data by controlling which database fields are published by the stylesheet
- hide metadata, such as database and field names
- format how the data is presented in a web page and control how the web user interacts with the data
- output the data as HTML or text, such as vCards or comma-separated values
- transform the data from a FileMaker XML grammar to a different XML grammar for use in another database or application, such as Scalable Vector Graphics (SVG)
- integrate any subset of the FileMaker data into other websites and with other middleware and custom applications that are potentially very different from the FileMaker database
- change the published field names to prevent unauthorized use of the database design information

Note Custom Web Publishing with XSLT for FileMaker Server is based on the W3C recommendation for XSLT 1.0. For information on XSLT 1.0, see www.w3.org. Additional functionality such as session management, email sending, and access to cookies and headers is provided by FileMaker XSLT extension functions. For information, see “Using the FileMaker XSLT extension functions and parameters” on page 59. The Web Publishing Engine does not support XSL Formatting Objects (XSL-FO).

What are some examples of using FileMaker XSLT stylesheets?

Here are just a few of the many possible examples of using FileMaker XSLT stylesheets:

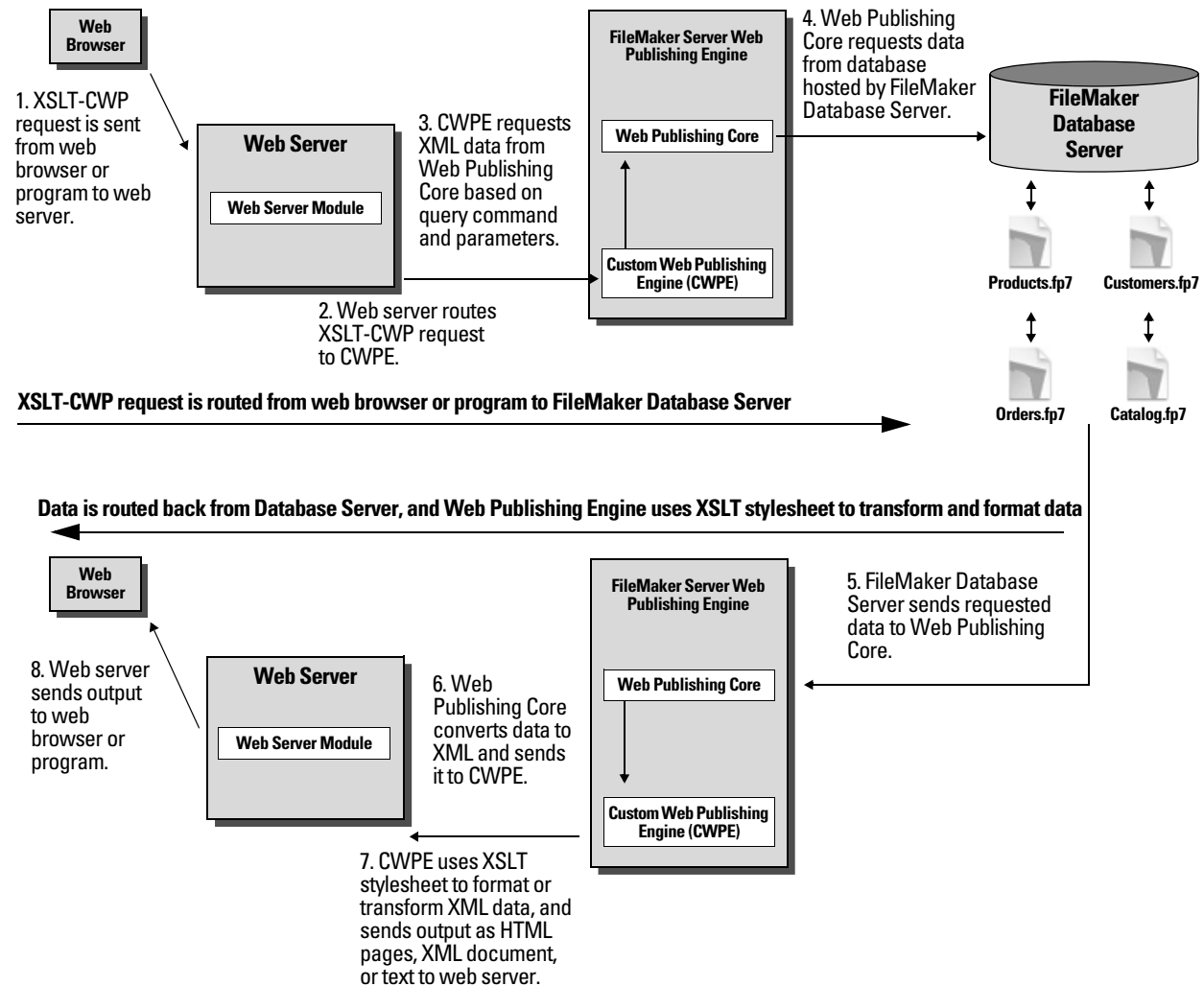
- You can insert a table in a web page for web users to browse that contains a subset of the data from a FileMaker database. For example, the table might contain people’s names and addresses, but not their phone numbers. To prevent unauthorized access, the web page can show generic labels for the data (such as “Name”) instead of the actual field names in the FileMaker database, such as “first_name.”
- You can create a web page or application that integrates data from a FileMaker portal with information from other data sources.
- You can add a button on a web page that creates a vCard from a person’s contact information in a FileMaker database.
- You can transform the XML data from a FileMaker database into an XML grammar that a spreadsheet or database application can open.

Getting started using Custom Web Publishing with XSLT

If you know standard XML and XSLT, then you can immediately start using the Web Publishing Engine after learning a few unique details about FileMaker XML and XSLT publishing, such as how to use the FileMaker XSLT extension functions and query commands and parameters. XSLT Site Assistant is a tool to help you create stylesheets and learn how they are constructed. Then, you can use your favorite XML and XSLT authoring tools to further enhance the stylesheets.

How the Web Publishing Engine generates pages based on XML data and XSLT stylesheets

After an XSLT Custom Web Publishing (XSLT-CWP) request is sent to the web server, the Web Publishing Engine queries the FileMaker database based on the query commands and parameters that are defined in the stylesheet and in the URL, and then outputs the data according to the instructions in the XSLT stylesheet.



General steps for using Custom Web Publishing with XSLT

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

1. In the Admin Console, make sure XSLT Publishing is enabled. See FileMaker Server Help.
2. Using FileMaker Pro, open each FileMaker database that you're publishing and make sure the database has the `fmxsft` extended privilege enabled for Custom Web Publishing with XSLT. See "Enabling Custom Web Publishing in a database" on page 19.

Note Make sure that you use equivalent FileMaker database privilege sets when developing stylesheets 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. Create XSLT stylesheets that include FileMaker-specific XSLT extension functions, query commands, and query parameters to format or transform the XML data from a FileMaker database.

You can use FileMaker XSLT Site Assistant to create one or more basic XSLT stylesheets as a starting point for your site. See the next section, "Using FileMaker XSLT Site Assistant to generate FileMaker XSLT stylesheets."

You can also use your own XSLT authoring or text editing tools to modify the XSLT stylesheets as necessary, or to develop your stylesheets from scratch. See chapter 6, "Developing FileMaker XSLT stylesheets."

4. Copy or place the XSLT stylesheets in the `xslt-template-files` folder, which is located inside the Web Publishing folder inside the FileMaker Server folder on the host where the Web Publishing Engine is installed.

You can also place the stylesheets in an optional folder or folder hierarchy inside the `xslt-template-files` folder.

5. Place any static files on the web server. See "Using FileMaker XSLT stylesheets in a website or program" on page 30.

6. Create or modify a website or program that uses the XSLT stylesheets.

For example, you can use a static page such as `index.html` for the website that auto-forwards web users to an XSLT stylesheet, or has a link to the XSLT stylesheet.

7. Make sure that security mechanisms for your site or program are in place.
8. Test the site or program with the XSLT stylesheets, using the same accounts and privileges that are defined for web users.
9. Make the site or program available and known to users.

Using FileMaker XSLT Site Assistant to generate FileMaker XSLT stylesheets

FileMaker XSLT Site Assistant is an application you can use to create basic XSLT stylesheets as a starting point for use with Custom Web Publishing with XSLT. XSLT Site Assistant is a good way to learn how FileMaker XSLT stylesheets are constructed. You can then use your own XSLT stylesheet authoring or text editing tools to modify the stylesheets as necessary. You cannot use XSLT Site Assistant to edit or update existing stylesheets, but you can use XSLT Site Assistant to generate the initial stylesheets for an entire site, or a single stylesheet to add basic functionality (such as deleting records) to an existing site.

You can use XSLT Site Assistant to generate XSLT stylesheets for all of the types of pages that are useful for working with FileMaker databases via Custom Web Publishing. Depending on the options you choose in XSLT Site Assistant, you can create a site that allows users to:

- browse a single record at a time
- view a list of all the records in the database
- search the database and view the results in a list
- sort records
- add records
- edit and duplicate records
- delete records
- view a summary report

You can also generate an optional home page that is linked to the other generated XSLT stylesheet pages.

The Web Publishing Engine uses each of your stylesheets to dynamically obtain data from a FileMaker database whenever a web user sends an HTTP request and a URL that references one of your XSLT stylesheets. The Web Publishing Engine uses a stylesheet to transform and format the XML data, and generates the resulting HTML page that the web user can work with.

Note The XSLT Site Assistant stylesheets transform FileMaker XML data into HTML pages based on the `fmresultset` XML grammar, which makes the stylesheets incompatible with other uses of XML data such as FileMaker Pro XML export.

Before using XSLT Site Assistant

Before you can use XSLT Site Assistant to generate XSLT stylesheets for a database:

- Set the extended privilege `fmxml` in the database. Use privilege sets when running XSLT Site Assistant that are equivalent to those you give to web users. See “Enabling Custom Web Publishing in a database” on page 19.
- Open and host the database on the Database Server component of FileMaker Server. See FileMaker Server Help.
- Be sure the web server component of the FileMaker Server deployment is running.
- Be sure the Web Publishing Engine component of the FileMaker server deployment is running.
- Enable XSLT Publishing in the Web Publishing Engine for using and testing the XSLT stylesheets. See FileMaker Server Help.

Starting XSLT Site Assistant

Note To use XSLT Site Assistant, you must have Java Runtime Environment 5 or Java Runtime Environment 6 installed.

To start XSLT Site Assistant:

1. Open a browser to the FileMaker Server Web Publishing Tools page.

Go to the following URL:

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

Where *<server>* is the machine on which the FileMaker Server resides.

2. Click PHP Site Assistant and XSLT Site Assistant Tools to go to the FileMaker Server Web Publishing Tools page.

3. Click Start XSLT Site Assistant.

FileMaker Server installs the required JAR files on your local machine, displaying a progress dialog until the procedure is complete.

4. (Optional) After the files are installed, you may select whether to install an icon for XSLT Site Assistant on your desktop. Click **OK** to install the icon.

You can now begin using XSLT Site Assistant.

Using XSLT Site Assistant

For detailed information and step-by-step procedures for using XSLT Site Assistant, see XSLT Site Assistant Help. For information about using the XSLT Site Assistant's generated stylesheets, see "Using FileMaker XSLT stylesheets in a website or program" on page 30.

Important When using XSLT Site Assistant, if you select a database that contains multiple tables, be sure to select layouts that are associated with the same table or else the generated site will return unexpected results. For example, a database might contain a Products table and a Customers table. When you select the layouts for a search page, an edit records page, and an add records page, be sure the layouts are all associated with the same table.

About XSLT Site Assistant's generated stylesheets

The XSLT stylesheets generated by XSLT Site Assistant include several FileMaker-specific processing instructions, elements, and parameters. Here are a few examples of what is included:

- The `<?xslt-cwp-query params="query string-fragment"?` processing instruction specifies the XML grammar to use and statically defines the name of the database you chose in XSLT Site Assistant. See "Using statically defined query commands and query parameters" on page 55.

- The `<xsl:param name="request-query"/>` element is used to access query information in a request or HTML form data. For example, this element can be used in the XSLT Site Assistant stylesheets to access the current request query information to determine the current location in a found set of records and to create links to the previous and next record. See “Accessing the query information in a request” on page 60.
- The `<xsl:param name="authenticated-xml-base-uri"/>` element, which isn’t always included, is used to access the authenticated base URI in a request when more XML data is needed within the request. See “Using the authenticated base URI parameter” on page 61.

XSLT Site Assistant also generates the `utilities.xsl` stylesheet for defining errors and common XSLT templates that are called by several XSLT Site Assistant stylesheets.

For information about other sections of the XSLT Site Assistant stylesheets, see chapter 6, “Developing FileMaker XSLT stylesheets.”

Using FileMaker XSLT stylesheets in a website or program

Whether you have used XSLT Site Assistant to generate XSLT stylesheets, or you have created your own stylesheets from scratch, the steps for using them in a website or program with the Web Publishing Engine are the same.

To use FileMaker XSLT stylesheets in a website or program:

1. Copy or place the XSLT stylesheets in the `xslt-template-files` folder, which is located inside the **Web Publishing** folder inside the **FileMaker Server** folder on the host where the Web Publishing Engine is installed.
You can also place the stylesheets in an optional folder or folder hierarchy inside the `xslt-template-files` folder.
2. If your XSLT stylesheets reference static files, such as static images or HTML files, place the static files in their original folder hierarchy within the root folder on the web server. Make sure the relative path is preserved.
For example, suppose an XSLT stylesheet references an image file called `logo.jpg` by using the HTML tag ``. The `logo.jpg` file must be located in the following location on the web server:
`<root folder>/fmi/xsl/logo.jpg`
3. 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, 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 21.

Note If the container fields store 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.

4. To request and process an XSLT stylesheet, use the following URL syntax:

```
<scheme>://<host>[:<port>]/fmi/xsl/<folder>/<stylesheet>.xsl[?<query string>]
```

See “About the URL syntax for FileMaker XSLT stylesheets” on page 52.

Note For websites, it is a good practice to include an XSLT stylesheet as a home page that doesn’t require users to enter a query string to access it. XSLT Site Assistant can create a `home.xml` file that doesn’t require a query string because it uses the `<?xslt-cwp-query?>` processing instruction. For example, if you copied your stylesheets (including a `home.xml` stylesheet) into the `my_templates` folder inside the `xslt-template-files` folder, web users can use the following URL to request and process the stylesheets:

```
http://192.168.123.101/fmi/xsl/my_templates/home.xml
```

Important The Web Publishing Engine does not allow web users to view the source for XSLT stylesheets that are installed in the `xslt-template-files` folder. When web users send a request to process a stylesheet, the Web Publishing Engine only sends the result of the stylesheet transformation to the web browser or program.

Troubleshooting XSLT stylesheets

If you have trouble using the XSLT stylesheets, verify that:

- The extended privileges in the database are set for Custom Web Publishing with XSLT and assigned to a user account. See “Enabling Custom Web Publishing in a database” on page 19.
- The database is hosted and opened on the Database Server component of 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.
- XSLT Publishing is enabled in the Web Publishing Engine.
 - Open the FileMaker Server Technology Tests page in a browser:


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

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

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

Chapter 5

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. In the same way that HTML has become the standard display language for communication on the World Wide Web, XML has become the standard language for structured data interchange. 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 immediately 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.

You can also use the Web Publishing Engine's server-side XSLT stylesheets to filter the XML data, reformat the data into HTML or text such as vCards, or transform the data into other XML grammars such as Scalable Vector Graphics (SVG). See chapter 4, "Introduction to Custom Web Publishing with XSLT" and chapter 6, "Developing FileMaker XSLT stylesheets."

For more general information on XML, additional examples that use XML, and links to XML resources, see the FileMaker website at www.filemaker.com.

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, which is available 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:

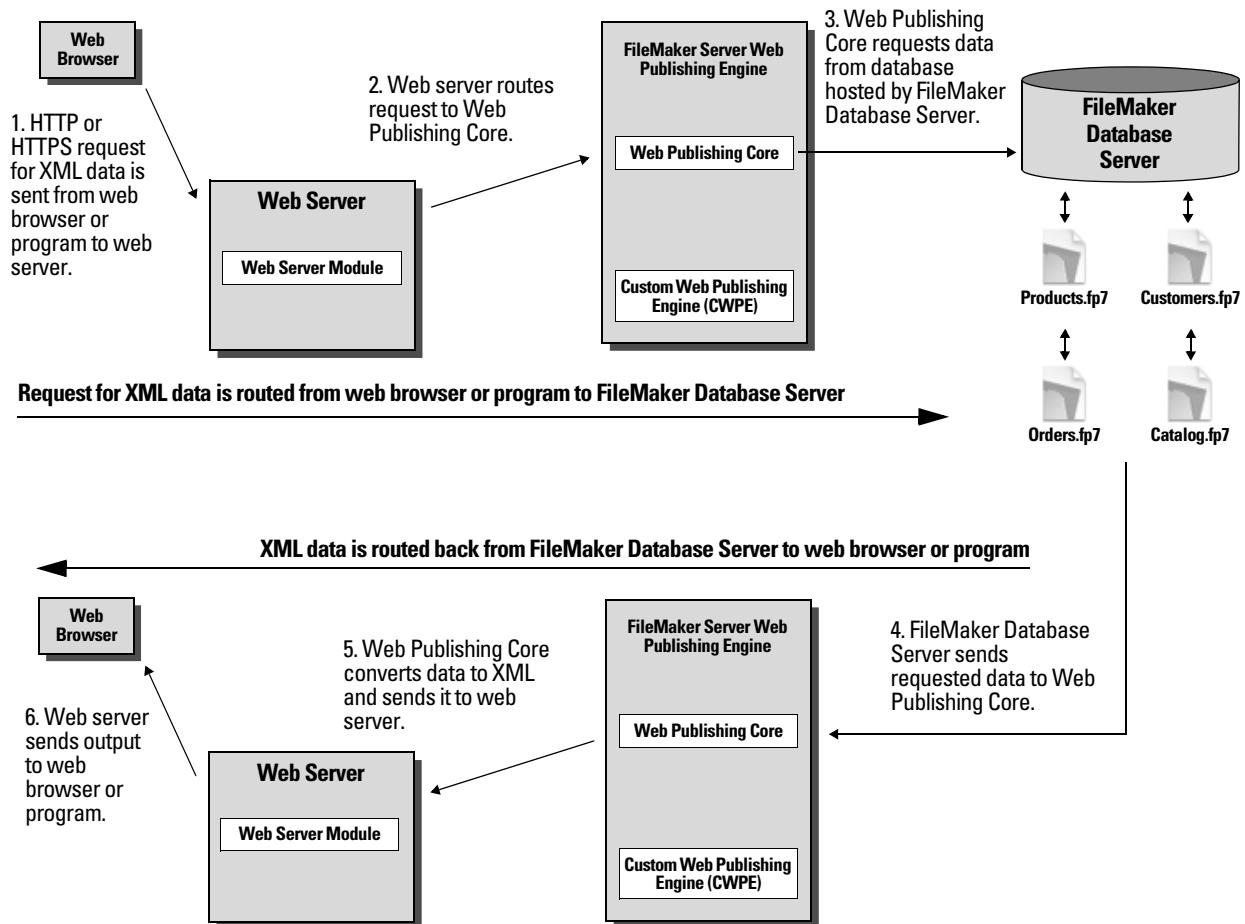
- For accessing XML data and XSLT web publishing, the Web Publishing Engine supports the `fmresultset`, `FMPXMLRESULT`, and `FMPXMLLAYOUT` grammars. For XML import, FileMaker Pro uses the `FMPXMLRESULT` grammar, and for export, FileMaker Pro uses the `FMPXMLRESULT` or `FMPDSORESET` grammar. See "Accessing XML data via the Web Publishing Engine" on page 37.
- 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 `fxml` extended privilege enabled for XML Custom Web Publishing. See "Enabling Custom Web Publishing in a database" on page 19.

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 order), 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 web page. 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 46, and appendix A, "Valid names used in 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.

If you specified a client-side stylesheet, the web browser parser also applies the stylesheet instructions. See "Using server-side and client-side processing of stylesheets" on page 49.

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. The URL syntax for using XSLT stylesheets is different from XML. See "About the URL syntax for FileMaker XSLT stylesheets" on page 52 and "About the URL syntax for FileMaker container objects in XSLT solutions" on page 53.

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.xml`, `FMPXMLRESULT.xml`, `FMPXMLLAYOUT.xml`, or `FMPDSORESULT.xml`. See “Using the `fmsresultset` grammar” on page 39 and “Using other FileMaker XML grammars” on page 42.
- `<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 46, and appendix A, “Valid names used in 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 three uppercase grammar names: `FMPXMLRESULT`, `FMPXMLLAYOUT`, and `FMPDSORESULT`. For information on the rules for case sensitivity of the query string, see “Guidelines for using query commands and parameters” on page 88.

Here are two examples of URLs for accessing XML data via the Web Publishing Engine:

```
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, then 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.”

For 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 89.

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.”

For 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, then the container field's <data> element contains a relative path that refers to the object. For 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 21.

- If a container field is empty, then the container field's <data> element is empty.

Note The syntax for container objects using XML is different from the syntax for container objects using XSLT. See “About the URL syntax for FileMaker container objects in XSLT solutions” on page 53.

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 46.

For 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
```

In this example URL, %C3%A9 is the URL encoded UTF-8 representation of the é character.

For more information on URL text encoding, see the URL specification, which is available 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. It is flexible and optimized for XSLT stylesheet authoring with easier field access by name and 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. You can use this grammar for accessing XML data and for XSLT stylesheets. To facilitate web publishing, this grammar is designed to be more verbose than the FMPXMLRESULT grammar. See “Using the fmsresultset grammar” on page 39.
- **FMPXMLRESULT and FMPXMLLAYOUT:** You can also use the FMPXMLRESULT and FMPXMLLAYOUT grammars with the Web Publishing Engine for accessing XML data and for XSLT stylesheets. 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 42.
- **FMPDSORESLT:** The FMPDSORESLT grammar, which is supported in FileMaker Pro for exporting XML, is deprecated for accessing XML data via the Web Publishing Engine. The FMPDSORESLT grammar is not supported for XSLT stylesheets. For information on the FMPDSORESLT grammar, see FileMaker Pro Help.

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 web page 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 46.

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>
FMPDSORESET	<ERRORCODE>0</ERRORCODE>

The error code element in the XML document indicates errors related to the database and query strings. Other types of errors can also occur for XSLT stylesheets and are handled differently. See appendix B, “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
FMPDSORESET	http://<host>[:<port>]/fmi/xml/FMPDSORESET.dtd?–db=<database>&–lay=<layout>

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

For 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 (00 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.

Example of XML data in the fmresultset grammar

The following is an example of XML data generated with the fmresultset grammar.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE fmresultset PUBLIC "-//FMI//DTD fmresultset//EN" ""http://localhost:16014/fmi/xml/fmresultset.dtd">
<fmresultset xmlns="http://www.filemaker.com/xml/fmresultset" version="1.0">
<error code="0" />
<product build="12/31/2012" name="FileMaker Web Publishing Engine" version="0.0.0.0" />
<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" />
<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>
<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" />
</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>
<relatedset count="0" table="artlocations" />
<field name="Style">
<data />
</field>
<field name="length">
<data>19</data>
</field>
</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

For 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

For 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 the tables in “Description of elements in the `fmresultset` grammar” on page 39.

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 99), 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.

Example of XML data in the FMPXMLRESULT grammar

The following is an example of XML data generated with the FMPXMLRESULT grammar.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FMPXMLRESULT PUBLIC "-//FMI//DTD FMPXMLRESULT//EN" ""http://localhost:16014/fmi/xml/
FMPXMLRESULT.dtd">
<FMPXMLRESULT xmlns="http://www.filemaker.com/fmpxmlresult">
  <ERRORCODE>0</ERRORCODE>
  <PRODUCT BUILD="12/31/2012" NAME="FileMaker Web Publishing Engine" VERSION="0.0.0.0" />
  <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="6" RECORDID="15">
      <COL>
        <DATA>Spring in Giverny 4</DATA>
      </COL>
      <COL>
        <DATA>Claude Monet</DATA>
      </COL>
      <COL>
        <DATA>/fmi/xml/cnt/data.jpg?-db=art&-lay=web&-recid=15&-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. In the following XML data 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. In the following XML data 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. In the following XML data 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. See “Using the date, time, and day extension functions” on page 72. 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.

Example of XML data in the FMPXMMLAYOUT grammar

The following is an example of XML data generated with the FMPXMMLAYOUT grammar.

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FMPXMMLAYOUT PUBLIC "-//FMI//DTD FMPXMMLAYOUT//EN" ""http://localhost:16014/fmi/xml/
FMPXMMLAYOUT.dtd">
<FMPXMMLAYOUT xmlns="http://www.filemaker.com/fmpxmmlayout">
  <ERRORCODE>0</ERRORCODE>
  <PRODUCT BUILD="12/31/2012" NAME="FileMaker Web Publishing Engine" VERSION="0.0.0.0" />
  <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="Impressionism">100</VALUE>
      <VALUE DISPLAY="Cubism">101</VALUE>
      <VALUE DISPLAY="Abstract">102</VALUE>
    </VALUELIST>
  </VALUELISTS>
</FMPXMMLAYOUT>
```

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. For 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 or in a `<?xslt-cwp-query?>` processing instruction in a FileMaker XSLT stylesheet. See chapter 6, “Developing FileMaker XSLT stylesheets.”

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

Note The Web Publishing Engine also supports an additional query command (`-process`) and three query parameters that are defined for use only with FileMaker XSLT stylesheets. See “Using query strings in FileMaker XSLT stylesheets” on page 54.

Use this query command name	To execute this command
<code>-dbnames</code>	Retrieve names of all hosted and web-shared databases.
<code>-delete</code>	Delete record.
<code>-dup</code>	Duplicate record.
<code>-edit</code>	Edit record.
<code>-find</code>	Find record(s).
<code>-findall</code>	Find all records.

Use this query command name	To execute this command
<code>-findany</code>	Find a random record.
<code>-findquery</code>	Perform complex or compound find request.
<code>-layoutnames</code>	Retrieve names of all available layouts for a hosted and web-shared database.
<code>-new</code>	Add new record.
<code>-scriptnames</code>	Retrieve names of all available scripts for a hosted and web-shared database.
<code>-view</code>	Retrieves layout information from a database if the <code>FMPXMLLAYOUT</code> grammar is specified. Retrieves <code><metadata></code> section of XML document and an empty recordset if the <code>fmresultset</code> or <code>FMPXMLRESULT</code> grammar is specified.

Use these query parameter names	With these query commands
<code>-db</code> (database name)	Required with all query commands except <code>-dbnames</code> and <code>-process</code> (XSLT requests only)
<code>-delete.related</code>	Optional with <code>-edit</code>
<code>-field</code>	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 36.
<code>fieldname</code>	At least one field name is required with <code>-edit</code> . Optional with <code>-find</code> . See “ <code>fieldname</code> (Non-container field name) query parameter” on page 96.
<code>fieldname.op</code> (operator)	Optional with <code>-find</code>
<code>-lay</code> (layout name)	Required with all query commands, except <code>-dbnames</code> , <code>-layoutnames</code> , <code>-scriptnames</code> , and <code>-process</code> (XSLT requests only)
<code>-lay.response</code> (switch layout for XML response)	Optional with all query commands, except <code>-dbnames</code> , <code>-layoutnames</code> , <code>-scriptnames</code> , and <code>-process</code> (XSLT requests only)
<code>-lop</code> (logical operator)	Optional with <code>-find</code>
<code>-max</code> (maximum records)	Optional with <code>-find</code> , <code>-findall</code>
<code>-modid</code> (modification ID)	Optional with <code>-edit</code>
<code>-query</code>	Required with <code>-findquery</code> compound find requests.
<code>-recid</code> (record ID)	Required with <code>-edit</code> , <code>-delete</code> , <code>-dup</code> . Optional with <code>-find</code>
<code>-relatedsets.filter</code>	Optional with <code>-find</code> , <code>-edit</code> , <code>-new</code> , <code>-dup</code> , and <code>-findquery</code> .
<code>-relatedsets.max</code>	Optional with <code>-find</code> , <code>-edit</code> , <code>-new</code> , <code>-dup</code> , and <code>-findquery</code> .
<code>-script</code> (perform script)	Optional with <code>-find</code> , <code>-findall</code> , <code>-findany</code> , <code>-new</code> , <code>-edit</code> , <code>-delete</code> , <code>-dup</code> , <code>-view</code>
<code>-script.param</code> (pass a parameter value to the script specified by <code>-script</code>)	Optional with <code>-script</code>
<code>-script.prefind</code> (perform script before <code>-find</code> , <code>-findany</code> , and <code>-findall</code>)	Optional with <code>-find</code> , <code>-findany</code> , <code>-findall</code>
<code>-script.prefind.param</code> (pass a parameter value to the script specified by <code>-script.prefind</code>)	Optional with <code>-script.prefind</code>
<code>-script.presort</code> (perform script before sort)	Optional with <code>-find</code> , <code>-findall</code>
<code>-script.presort.param</code> (pass a parameter value to the script specified by <code>-script.presort</code>)	Optional with <code>-script.presort</code>
<code>-skip</code> (skip records)	Optional with <code>-find</code> , <code>-findall</code>
<code>-sortfield.[1-9]</code> (sort field)	Optional with <code>-find</code> , <code>-findall</code>
<code>-sortorder.[1-9]</code> (sort order)	Optional with <code>-find</code> , <code>-findall</code>

Use these query parameter names	With these query commands
<code>-stylehref</code> (stylesheet HREF)	Optional with all query commands (to specify a stylesheet URL for <code>-styletype</code>)
<code>-styletype</code> (stylesheet type)	Optional with all query commands (to specify client-side stylesheet)

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.

For 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 `-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 `-lay.response` query parameter to switch to a different layout, if this is specified.
8. Process the `-script` query parameter, if 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 non-existent field, the request deletes the current record and returns error code 102 (“Field is missing”), but does not execute the script.

Using server-side and client-side processing of stylesheets

The Web Publishing Engine supports server-side processing of an XSLT stylesheet, and also allows you to use a query parameter that specifies client-side processing of a stylesheet.

It is important to understand the differences between the two ways to process stylesheets, and the security implications of using client-side processing. Server-side processing is more secure than client-side processing because server-side processing does not give web users access to the unfiltered XML data. With server-side processing, the data is presented in a form that the data owner or the XSLT stylesheet author decides is appropriate to present. Server-side processing hides the database names, field names, and other implementation details from web users. Server-side processing can also be used to specify statically defined query parameters, which prevent the use of unauthorized query commands and query parameters, such as database names. See chapter 4, “Introduction to Custom Web Publishing with XSLT” and chapter 6, “Developing FileMaker XSLT stylesheets.”

If your solution requires client-side stylesheet processing, you can have the Web Publishing Engine generate an XML stylesheet processing instruction with each grammar by including the `-styletype` and `-stylehref` parameters in the FileMaker query string request. You can use cascading stylesheets (CSS) or XSLT stylesheets for displaying your XML document.

- The `-styletype` parameter is used for setting the value of the type attribute (`type=text/css` or `type=text/xml`).
- The `-stylehref` parameter is used for setting the value of the HREF attribute that specifies the location of the stylesheet using an absolute path. For example: `href=/mystylesheet.css` or `href=/stylesheets/mystylesheet.xml`. The name of the stylesheet can be any name but it must contain an extension of either `.css` or `.xml`.

Here is an example of a FileMaker query string that generates client-side stylesheet processing:

```
http://localhost/fmi/xml/fmresultset.xml?-db=products-lay=sales&-findall&-styletype=text/xml&-stylehref=/mystylesheet.xml
```

Note This “/” in “`-stylehref=/document.xml`” in this example is used because the stylesheet is located in the root folder of the web server software. Use a URL for the stylesheet that uses an absolute path to specify its location on the web server. The stylesheet can also be located on another web server.

Based on this request, the Web Publishing Engine will include the following processing instruction in the XML document:

```
<?xml-stylesheet type="text/xml" href="/mystylesheet.xml"?>
```

Copy or place the stylesheet for client-side processing on the web server in the location specified by the absolute path in the URL for the HREF attribute.

Important Do not place stylesheets for client-side processing inside the `xslt-template-files` folder, which is used for server-side processing of XSLT stylesheets. See “Using FileMaker XSLT stylesheets in a website or program” on page 30.

Note Some web browsers do not support client-side processing. For information, see the documentation for your web browser.

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 19.
- 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 6

Developing FileMaker XSLT stylesheets

This chapter contains information about how FileMaker XSLT stylesheets are constructed and how to use the FileMaker XSLT extension functions.

Using XSLT stylesheets with the Web Publishing Engine

When developing and using XSLT stylesheets to request FileMaker XML data via the Web Publishing Engine, be aware of the following points:

- To use an XSLT stylesheet with the Web Publishing Engine, you must specify the name of the XSLT stylesheet in a URL. If you don't specify a stylesheet, or if the Web Publishing Engine is unable to find or parse the stylesheet, the Web Publishing Engine displays an error page. See "About the URL syntax for FileMaker XSLT stylesheets" on page 52.
- The stylesheet filename and the folder name where the stylesheet is stored must be UTF-8 URL-encoded. If your stylesheet must be compatible with older web browsers, limit the names to ASCII characters.
- You must specify the FileMaker XML grammar to use, either as a query parameter in the URL, or as a statically defined query parameter in the `<?xslt-cwp-query?>` processing instruction. If you don't specify an XML grammar, the Web Publishing Engine displays an error. See "Specifying an XML grammar for a FileMaker XSLT stylesheet" on page 54.
- You can specify the query parameters that identify the FileMaker XML data you want to request either in the URL, or as a statically defined query parameter in the `<?xslt-cwp-query?>` processing instruction. See "About the URL syntax for FileMaker XSLT stylesheets" on page 52 and "Using statically defined query commands and query parameters" on page 55.
- You can optionally specify the text encoding of an XSLT request by using the `-encoding` query parameter. If you don't specify an encoding, the Web Publishing Engine uses its default text encoding setting for requests. See "Setting text encoding for requests" on page 57.
- You can optionally specify an output method via the `method` attribute of the `<xsl:output>` element. If you don't specify an output method, the Web Publishing Engine uses HTML as the output. You can also optionally specify the output page encoding by using the `encoding` attribute of the `<xsl:output>` element. If you don't specify an encoding, the Web Publishing Engine uses the default text encoding setting for output pages. See "Specifying an output method and encoding" on page 58.
- You can optionally specify the text encoding for email messages sent from the Web Publishing Engine via a function parameter for the `fmxslt:send_email()` extension function. See "Sending email messages from the Web Publishing Engine" on page 66.

To construct a request, the Web Publishing Engine begins by using any query command and query parameters that are statically defined in the optional `<?xslt-cwp-query?>` processing instruction. The statically defined query command and parameters become the base request. The `<?xslt-cwp-query?>` processing instruction is not required in a stylesheet, but its base request takes precedence over any matching query command or parameters that are specified in the URL query string. The Web Publishing Engine then adds to the base request any query command or additional parameters in the URL query string that are *not* defined in the `<?xslt-cwp-query?>` processing instruction. The Web Publishing Engine uses this request to obtain the FileMaker XML data and return it to your web browser or program in the output method you specified, or as HTML.

About the FileMaker XSLT Extension Function Reference

This release includes a FileMaker database called XSLT Reference.fp7 that contains brief descriptions and examples of each of the FileMaker XSLT extension functions. The function reference database can be found in the following directory on any machine in your FileMaker Server deployment (master or worker).

Mac OS

/Library/FileMaker Server/Example/XSLT

Windows

<drive>:\Program Files\FileMaker\FileMaker Server\Examples\XSLT

where <drive> is the primary drive from which the system is started.

About the FileMaker XSLT Starter Solution

This release includes a FileMaker XSLT starter solution that provides an example of what you can do with XSLT solutions. The XSLT starter solution can be found in the following directory on any machine in your FileMaker Server deployment (master or worker).

Mac OS

/Library/FileMaker Server/Example/XSLT/Starter Solution

Windows

<drive>:\Program Files\FileMaker\FileMaker Server\Examples\XSLT\Starter Solution

where <drive> is the primary drive from which the system is started.

About the URL syntax for FileMaker XSLT stylesheets

The URL syntax for using FileMaker XSLT stylesheets with the Web Publishing Engine is:

<scheme>://<host>[:<port>]/fmi/xsl/[<path>]/<stylesheet.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).
- <path> is optional and specifies the folder(s) inside the xslt-template-files folder where the XSLT stylesheet is located.

- `<stylesheet.xml>` is the XSLT stylesheet filename.
- `<query string>` can be a combination of one query command and one or more query parameters for Custom Web Publishing with XSLT. See “Using query strings in FileMaker XSLT stylesheets” on page 54, and appendix A, “Valid names used in query strings.” If the specified stylesheet includes a `<?xslt-cwp-query?>` processing instruction, the statically defined query command and parameters take precedence over any matching query command or parameters in the URL query string. See “Using statically defined query commands and query parameters” on page 55.

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 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 88.

Here is an example of a URL for using a FileMaker XSLT stylesheet with the Web Publishing Engine:

```
http://192.168.123.101/fmi/xsl/my_template/my_stylesheet.xml?–grammar=fmresultset&–db=mydatabase
&–lay=mylayout&–findall
```

About the URL syntax for FileMaker container objects in XSLT solutions

In a generated XML document for an XSLT 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, then the container field’s `<data>` element uses the following URL syntax to refer to the object:

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

where `<extension>` is the filename extension identifying the type of object, such as `.jpg` or `.mov`. For information on `<query string>`, see the previous section, “About the URL syntax for FileMaker XSLT stylesheets.”

For example:

```
<data>/fmi/xsl/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 89.

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

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

For information about `<scheme>`, `<host>`, or `<port>`, see the previous section, “About the URL syntax for FileMaker XSLT stylesheets.”

For example:

```
http://www.company.com/fmi/xsl/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, then the container field’s <data> element contains a relative path that refers to the object. For example, if logo.jpg was in the Web folder inside the FileMaker Pro folder, the container field’s <data> element is:

```
<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 21.

- If a container field is empty, then the container field’s <data> element is empty.

Using query strings in FileMaker XSLT stylesheets

When using a query string in a URL or in the <?xslt-cwp-query?> processing instruction in a FileMaker XSLT stylesheet, you can include any of the query commands and parameters that are defined for requesting XML data from a FileMaker database. See “Using FileMaker query strings to request XML data” on page 46.

You can also use the following query command and parameters that are defined for use only with FileMaker XSLT stylesheets.

Use this XSLT query command or parameter name

To	Comment
<code>-grammar3</code>	Specify the XML grammar for XSLT-CWP requests or for XSLT stylesheets. See the next section, “Specifying an XML grammar for a FileMaker XSLT stylesheet.” This query parameter is required in all XSLT requests.
<code>-encoding</code>	Specify the text encoding for a request. See “Setting text encoding for requests” on page 57. This query parameter is optional in all XSLT requests.
<code>-process</code>	Process a stylesheet without requesting data. See “Processing XSLT requests that do not query FileMaker Server” on page 58. This query command requires the <code>-grammar</code> query parameter.
<code>-token</code>	Pass values between pages without using sessions or cookies. See “Using tokens to pass information between stylesheets” on page 59. This query parameter is optional in all XSLT requests.

Specifying an XML grammar for a FileMaker XSLT stylesheet

The recommended XML grammar to use with Custom Web Publishing with XSLT is the `fmresultset` grammar, which has been designed for ease of use with XSLT. See “Using the `fmsresultset` grammar” on page 39. You can also use the older `FMPXMLRESULT` or `FMPXMMLAYOUT` grammars. To access value lists and field display information in layouts, you must use the `FMPXMMLAYOUT` grammar. See “Using other FileMaker XML grammars” on page 42. You cannot use the `FMPDSORESLT` grammar with Custom Web Publishing with XSLT.

To specify the grammar for a FileMaker XSLT stylesheet, use the `-grammar` query parameter in a URL or as a statically defined query parameter in the <?xslt-cwp-query?> processing instruction.

For example, in an URL:

```
http://192.168.123.101/fmi/xsl/my_template/my_stylesheet.xml?-grammar=fmresultset&-db=mydatabase
&-lay=mylayout&-findall
```

For example, in a processing instruction:

```
<?xslt-cwp-query params="-grammar=fmresultset&-db=mydatabase&-lay=mylayout&-findall"?>
```

Important If you don't specify an XML grammar for a FileMaker XSLT stylesheet, the error "QUERY - ER0001" is displayed. See appendix B, "Error codes for Custom Web Publishing."

About namespaces and prefixes for FileMaker XSLT stylesheets

Unique XSLT namespaces help distinguish XSLT tags by the application they were designed for. In the `<xsl:stylesheet>` element at the start of all FileMaker XSLT stylesheets, declare the namespaces for the FileMaker XSLT extension functions and the particular grammars you are using in the stylesheet.

If you use this	Declare this namespace	Use this prefix
fmresultset XML grammar	<code>xmlns:fmrs="http://www.filemaker.com/xml/fmresultset"</code>	fmrs
FMPXMLRESULT grammar	<code>xmlns:fmp="http://www.filemaker.com/fmpxmlresult"</code>	fmp
FMPXMLLAYOUT grammar	<code>xmlns:fml="http://www.filemaker.com/fmpxmllayout"</code>	fml
For the query XML grammar	<code>xmlns:fmq="http://www.filemaker.com/xml/query"</code>	fmq
For the FileMaker XSLT extension functions	<code>xmlns:fmxslt="xalan://com.fmi.xslt.ExtensionFunctions"</code>	fmxslt

You must also declare the following required namespace in each FileMaker XSLT stylesheet:

```
xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
```

Here is an example of namespace declarations:

```
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fmrs="http://www.filemaker.com/xml/fmresultset"
  xmlns:fml="http://www.filemaker.com/fmpxmllayout"
  xmlns:fmq="http://www.filemaker.com/xml/query"
  xmlns:fmxslt="xalan://com.fmi.xslt.ExtensionFunctions"
  exclude-result-prefixes="xsl fmrs fmq fml fmxslt">
```

Using statically defined query commands and query parameters

You can prevent the unauthorized use of query commands and query parameters with your FileMaker XSLT stylesheet by statically defining the query commands and parameters that you want to use when XML data is requested. Although not required, if any query commands and parameters are statically defined in a stylesheet, they take precedence over any matching query command or parameters that a client may attempt to specify in the URL query string.

The stylesheets generated by XSLT Site Assistant use statically defined query commands and parameters. FileMaker recommends using statically defined query commands and parameters as a best practice technique for enhancing the security of your solution.

To statically define query commands and parameters, use the following processing instruction at the beginning of your FileMaker XSLT stylesheet:

```
<?xslt-cwp-query params="query string-fragment"?>
```

where:

query string-fragment is a string that contains the name-value pairs in the following format:

```
name=value&name2=value2...
```

where:

name is a string that is the name of a query command, query parameter, or database field.

value is an arbitrary length string value. For query parameters and field names, use the particular value you want to define, such as “-db=products”. For query commands, don’t specify an equals (“=”) sign or a value after the command name, such as -findall. See appendix A, “Valid names used in query strings.”

The strings used in the fragment must be URL encoded. See “About URL text encoding” on page 37. You must use the same character encoding that is specified by the encoding attribute in the <xsl:output> tag. If no encoding is specified, then the Web Publishing Engine uses its configured default encoding.

The separator between two name value pairs must be an ampersand (&).

For example, suppose you used the following processing instruction in a stylesheet named my_stylesheet.xml:

```
<?xslt-cwp-query params="-db=products&-lay=sales&-grammar=fmresultset&productname=the%20item&-find"?>
```

This example processing instruction would force all requests for the my_stylesheet.xml to use the fmresultset grammar with the products database and the sales layout, and do a -find request with the productname field set to the value “the%20item”.

If a client made the following request using my_stylesheet.xml:

```
http://server.company.com/fmi/xsl/my_stylesheet.xml?-lay=revenue&city=London&-edit
```

then the Web Publishing Engine would process the following XML request:

```
http://server.company.com/fmi/xml/fmresultset.xml?-db=products&-lay=sales&productname=the%20item&city=London&-find
```

The statically defined query command and parameters override the -lay=revenue query parameter and the -edit query command provided by the client. Because the city field was not statically defined in the processing instruction, the Web Publishing Engine includes in the XML request the value of “London” for the city field that the client provided.

Setting text encoding for requests

The Web Publishing Engine performs the following steps in the order shown until it determines the encoding of an XSLT request:

1. Checks if the charset attribute is set in the Content-Type request header.
2. Checks if you specified an encoding with the `–encoding` query parameter. You can specify this parameter in a URL or as a statically defined query parameter in the `<?xslt-cwp-query?>` processing instruction. The value of the `–encoding` parameter indicates the encoding used on the rest of the parameters in the request. The valid values for this parameter are listed in the following table. For example:

```
http://192.168.123.101/fmi/xsl/template/my_stylesheer.xsl?–db=products–lay=sales&–grammar=fmresultset
&–encoding=Shift_JIS&–findall
```

3. Uses the current setting for the request and output pages default text encoding option for the Web Publishing Engine. When the Web Publishing Engine is first installed, the initial default text encoding setting for requests is UTF-8. You can change the Web Publishing Engine’s text encoding settings by using the Admin Console. See FileMaker Server Help.

After the Web Publishing Engine determines the encoding, that encoding is used and no further steps are taken to determine the encoding. For example, if the charset attribute is set in the Content-Type request header, the Web Publishing Engine does not use the value of the `–encoding` query parameter.

The text encoding that is specified via any of the methods above must use one of the following encodings:

Encoding	Description
US-ASCII	The basic ASCII character set that is typically used for plain text English email.
ISO-8859-1	The Latin 1 character set that is typically used for roman character based web pages and email messages requiring upper ASCII characters.
ISO-8859-15	The Latin 9 character set, which is almost the same as the Latin 1 character set with the addition of the Euro € symbol.
ISO-2022-JP	The ISO Japanese encoding that is typically used for Japanese email messages.
Shift_JIS	The Japanese encoding that is typically used for Japanese web pages.
UTF-8	The eight-bit encoding of Unicode. Using UTF-8 for email messages and web pages is growing in popularity as major browsers and email clients have added support. Because UTF-8 supports the full range of Unicode characters, it can handle pages for any language.

Notes

- When the Web Publishing Engine is first installed, the initial default text encoding setting for output pages is UTF-8. See the next section, “Specifying an output method and encoding.” For email messages, the Web Publishing Engine uses an initial default text encoding setting of ISO-8859-1. You can change these settings by using the Admin Console.
- You can also set email message encoding by using the `fmxml:send_email(String smtpFields, String body, String encoding)` extension function. See “Sending email messages from the Web Publishing Engine” on page 66.

Specifying an output method and encoding

You can specify an output method and encoding of output pages by using the method and encoding attributes of the `<xsl:output>` element. Both of these attributes are optional.

The method attribute specifies the type of output, which can be “html”, “text”, or “xml”. No other method types are supported. If you don’t specify a method, the Web Publishing Engine uses the “html” method.

The encoding attribute specifies the encoding of the output pages. You can specify any of the encodings listed in the table in the previous section. If you don’t specify an encoding, the Web Publishing Engine uses its default text encoding setting for output pages.

For example:

```
<xsl:output method="html" encoding="ISO-8859-1"/>
```

If you don’t use the `<xsl:output>` element in a stylesheet, the Web Publishing Engine outputs HTML pages using the current default text encoding setting for output pages.

About the encoding of XSLT stylesheets

In addition to the encoding for requests and output pages, the encoding of your XSLT stylesheets must be specified in the encoding attribute of the XML declaration at the top of the stylesheet. You can use any of the text encodings listed in the table on page 57.

For example, this declaration specifies UTF-8 as the encoding of the stylesheet:

```
<?xml version="1.0" encoding="UTF-8"?>
```

If you don’t specify the stylesheet encoding, the Web Publishing Engine assumes the encoding is UTF-8.

Processing XSLT requests that do not query FileMaker Server

You can use the `–process` query command to process XSLT requests that do not need any data from the database, or if your stylesheet does not require database-specific information, such as records, field names, or layout names. By using the `–process` command in these types of situations, you can reduce the workload for FileMaker Server.

For example, you can use the `–process` command to:

- load a stylesheet that generates a static page, if no database information is needed
- load a stylesheet that creates a new record, if the stylesheet does not require any database or layout information, such as a value list
- use an extension function such as `fmxml:send_email()` that doesn’t require data from the database
- access information stored in a session if no database information is needed

The `–process` command returns an XML document that contains product information about the Web Publishing Engine.

The only required parameter for the `–process` command is `–grammar`, and you must use the `fmresultset` grammar or the `FMPXMLRESULT` grammar.

For example:

```
http://192.168.123.101/fmi/xsl/my_template/my_stylesheet.xml?–grammar=fmresultset&–process
```

Using tokens to pass information between stylesheets

You can use the `-token` query parameter in a URL or as a statically defined query command to pass any user-defined information between stylesheets without using sessions or cookies. The `-token` query parameter is optional with all query commands.

The user-defined parameter value can be any character string that is URL encoded. For example:

```
http://192.168.123.101/fmi/xsl/template/my_stylesheet.xml?-db=products&-lay=sales&-grammar=fmresultset
&-token.D100=Pending&-findall
```

See “`-token.[string]` (Pass values between XSLT stylesheets) query parameter” on page 105.

Important Do not use the `-token` query parameter to pass private data.

To retrieve the value of the `-token` query parameter, use the `<xsl:param name="request-query" />` statement. See “Accessing the query information in a request” on page 60.

Using the FileMaker XSLT extension functions and parameters

The FileMaker XSLT extension functions are defined to be in the `fmxslt` namespace. Make sure you include a declaration of the `fmxslt` namespace in the `<xsl:stylesheet>` element at the start of your XSLT stylesheet. See “About namespaces and prefixes for FileMaker XSLT stylesheets” on page 55.

The FileMaker XSLT extension functions have been designed so that you can use them within an XSLT stylesheet by specifying them as a function call within an XPath statement. XPath statements are used as the values of the `select` attribute and the `test` attribute in numerous XSLT elements.

For example, suppose you want to check the User-Agent header to determine the browser being used. To do this, you might want to use a variable that contains the value of the User-Agent header:

```
<xsl:variable name="user-agent" select="fmxslt:get_header('User-Agent')"/>
```

For the extension functions that return a value, the value will be returned in the XSLT type specified. Many functions return strings, but a few functions return a `node-set` that can be traversed.

Note This section describes the FileMaker XSLT extension functions and parameters, and includes some examples. For additional examples of each function, see the FileMaker XSLT Extension Function Reference. See “About the FileMaker XSLT Extension Function Reference” on page 52.

About the FileMaker-specific XSLT parameters set by the Web Publishing Engine

When processing a request, the Web Publishing Engine dynamically sets the values of the following FileMaker-specific XSLT parameters. You can use the values of these parameters in your stylesheet by using the `<xsl:param>` element.

FileMaker-specific XSLT parameter	For more information, see
<code><xsl:param name="request-query"/></code>	“Accessing the query information in a request” in the next section.
<code><xsl:param name="client-ip"/></code> <code><xsl:param name="client-user-name"/></code> <code><xsl:param name="client-password"/></code>	“Obtaining client information” on page 60.
<code><xsl:param name="xml-base-uri"/></code>	“Using the Web Publishing Engine base URI parameter” on page 61.
<code><xsl:param name="authenticated-xml-base-uri"></code>	“Using the authenticated base URI parameter” on page 61.

Accessing the query information in a request

You can use a FileMaker XSLT parameter to access query information in a request in a URL or HTML form data. For example, you can access the current request query information to determine the current location in a found set of records, and create links to the previous and next record.

The following FileMaker XSLT parameter provides access to all of the query commands and query parameters that are used to request FileMaker XML data via the Web Publishing Engine:

```
<xsl:param name="request-query"/>
```

With the exception of field names, the Web Publishing Engine returns all query command and query parameter names in lowercase. The capitalization of field names is preserved.

An XML document fragment is loaded into the request-query parameter in the following grammar:

```
<!DOCTYPE query [
  <!ELEMENT query (parameter)*>
    <!ATTLIST query action CDATA #REQUIRED>
    <!ELEMENT parameter (#PCDATA)>
    <!ATTLIST parameter name CDATA #REQUIRED>
  ]
```

Note The query information is defined to be in the namespace `fmq="http://www.filemaker.com/xml/query"`. Make sure you include a declaration of the `fmq` namespace in the `<xsl:stylesheet>` element at the start of your XSLT stylesheet. See “About namespaces and prefixes for FileMaker XSLT stylesheets” on page 55.

For example, suppose you want to access the query commands and query parameters in this request:

```
http://192.168.123.101/fmi/xsl/my_stylesheet.xml?-db=products&-lay=sales&-grammar=fmresultset&-token.1=abc123
&-findall
```

If you include the `<xsl:param name="request-query" />` statement before the template section, the Web Publishing Engine will store this XML document fragment in that parameter:

```
<query action="my_stylesheet.xml" xmlns="http://www.filemaker.com/xml/query">
  <parameter name="-db">products</parameter>
  <parameter name="-lay">sales</parameter>
  <parameter name="-grammar">fmresultset</parameter>
  <parameter name="-token.1">abc123</parameter>
  <parameter name="-findall"></parameter>
</query>
```

You can then use the `request-query` parameter to access the value of a token that was passed in a URL by using an XPath expression. For example:

```
$request-query/fmq:query/fmq:parameter[@name = '-token.1']
```

Obtaining client information

You can use the following FileMaker XSLT parameters to obtain information from the Web Publishing Engine about a web client’s IP address, user name, and password:

```
<xsl:param name="client-ip"/>
<xsl:param name="client-user-name"/>
<xsl:param name="client-password">
```

Include these parameter statements in your XSLT stylesheet before the top `<xsl:template>` element.

These parameters provide the web user's credentials when a stylesheet programmatically loads additional password-protected XML documents. See "Loading additional documents" on page 61. The web user must provide the user name and password initially via the HTTP Basic Authentication dialog box. See "Accessing a protected database" on page 19.

For more information and examples of using these three FileMaker XSLT parameters, see the FileMaker XSLT Extension Function Reference.

Using the Web Publishing Engine base URI parameter

The Web Publishing Engine defines the base Uniform Resource Identifier (URI) parameter to be the host and port where the Web Publishing Engine is installed. The base URI allows requests for XML data from FileMaker databases to be resolved in relation to the Web Publishing Engine host.

To access the Web Publishing Engine base URI, include this statement in your XSLT stylesheet before the top `<xsl:template>` element:

```
<xsl:param name="xml-base-uri"/>
```

You can then use the base URI for the current stylesheet via the `$xml-base-uri` variable whenever you need to make an additional request for FileMaker XML data. For example, you can use the base URI in the following request for additional XML data:

```
<xsl:variable name="layout_information" select="document(concat($xml-base-uri,'fmi/xml/FMPXMMLAYOUT.xml?
-db=products&-lay=sales&-view'))" />
```

Using the authenticated base URI parameter

The `authenticated-xml-base-uri` parameter combines the functionality of the `client-user-name` and `client-password` parameters with the `xml-base-uri` parameter:

```
<xsl:param name="authenticated-xml-base-uri"/>
```

Use this parameter to load an additional password-protected XML document that requires the same user name and password that was specified in the original request currently being processed. For an example, see the next section, "Loading additional documents."

Include this parameter statement in your XSLT stylesheet before the top `<xsl:template>` element.

If the values for the `client-user-name` and `client-password` parameters are not blank, then the value of the `authenticated-xml-base-uri` parameter is:

```
http://username:password@hostname:port
```

If the values for the `client-user-name` and `client-password` parameters are blank, then the value of the `authenticated-xml-base-uri` parameter is the same as the value of the `xml-base-uri` parameter.

Loading additional documents

To load an additional XML document during the processing of an XSLT stylesheet, use the standard XSLT `document()` function with a URI to the XML document. The `document()` function returns the requested XML as a `node-set` that can be stored in an `<xsl:variable>` element.

To load an XML document that contains data from a FileMaker database, use the `document()` function with FileMaker query command and parameters. For example:

```
<xsl:variable name="other-data" select="document(concat($xml-base-uri,'fmi/xml/FMPXMMLAYOUT.xml?
-db=products&-lay=sales&-view'))"/>
```

To load an additional password-protected XML document that requires the same user name and password that was specified in the original request currently being processed, use the `authenticated-xml-base-uri` parameter. This parameter specifies the same user name and password as part of the URI that is passed to the `document()` function.

For example:

```
<xsl:variable name="other-data" select="document(concat($authenticated-xml-base-uri,
'/fmi/xml/FMPXMMLLAYOUT.xml?-db=products&-lay=sales&-view'))"/>
```

To load a password-protected XML document that requires a different user name and password than what was specified in the parent request, then use the following syntax to specify the user name and password as part of the URI that is passed to the `document()` function:

```
http://username:password@hostname/path?querystring
```

To load an XML document that is not based on a FileMaker database, use the `document()` function without FileMaker query commands or query parameters. For example:

```
<xsl:variable name="other-data" select="document('http://server.company.com/data.xml')" />
```

If you use the `document()` function with a relative URL, the Web Publishing Engine attempts to load the XML document from the local file system in the location relative to where the stylesheet is stored. For example, suppose a stylesheet that is located inside the `mystylesheets` folder inside the `xslt-template-files` folder contains the following `document()` function with a relative URL:

```
<xsl:variable name="mydoc" select="document('mystylesheets/mydoc.xml')" />
```

The Web Publishing Engine will attempt to load `mydoc.xml` from the `mystylesheets` folder inside the `xslt-template-files` folder in the local file system.

Note When you use the Web Publishing Engine's base URI to load a document, the Web Publishing Engine supports HTTP only. When you load a document from an external server, the Web Publishing Engine supports both HTTP and HTTPS.

Using the layout information for a database in a stylesheet

You can incorporate the layout information for a FileMaker database in a stylesheet by using the `FMPXMMLLAYOUT` grammar to request the information and then loading it into a variable via the XSLT `document()` function:

```
<xsl:variable name="layout" select="document(concat($xml-base-uri,'/fmi/xml/FMPXMMLLAYOUT.xml?-view'))" />
```

For example, suppose you wanted to create a pull down menu for a field named **Color** that is populated with the values from a two-field value list named **shirts** that is defined in a layout in a FileMaker database. Suppose the first field in the two-field value list stores the ID number of the color (such as “100”), and the second field stores the color’s associated name (such as “Light Green”). Here’s how you can use the `document()` function to load the layout information into a XSLT variable along with the `DISPLAY` attribute to display the value for the second field in a two-field value list:

```
<xsl:variable name="layout" select="document(concat($xml-base-uri,'/fmi/xml/FMPXMMLLAYOUT.xml?–db=products
&–lay=sales&–view'))" />
<select size="1">
  <xsl:attribute name="name">Color</xsl:attribute>
  <option value=""> Select One...</option>
  <xsl:for-each select="$layout/fmi:FMPXMMLLAYOUT/fmi:VALUELISTS/fmi:VALUELIST[@NAME = 'shirts']/
fmi:VALUE">
    <option>
      <xsl:attribute name="value"><xsl:value-of select="."/></xsl:attribute>
      <xsl:value-of select="@DISPLAY"/>
    </option>
  </xsl:for-each>
</select>
```

Using content buffering

When content buffering is disabled, the Web Publishing Engine streams the result of an XSLT transformation directly back to the client. Content buffering is always disabled unless you explicitly enable it. If you enable content buffering, the Web Publishing Engine stores the transformed content until the entire transformation is finished.

Content buffering is required for XSLT stylesheets that manipulate headers. Because headers are written before the response body, the body must be buffered so that the added header information can be included.

There are four FileMaker extension functions that require the XSLT transformation result to be buffered:

- `fmxslt:create_session()`: See “Using the session extension functions” on page 64.
- `fmxslt:set_header()`: See “Using the header functions” on page 67.
- `fmxslt:set_status_code ()`: See “Using the header functions” on page 67.
- `fmxslt:set_cookie()`: See “Using the cookie extension functions” on page 68.

In order for these FileMaker extension functions to work properly, you must include the following XSLT processing instruction in the top level document for the request:

```
<?xslt-cwp-buffer buffer-content="true"?>
```

Important If you have a base stylesheet that includes another stylesheet, then the base stylesheet must include the `<?xslt-cwp-buffer?>` processing instruction. This instruction is ignored if it is used in an included stylesheet.

A benefit of using the processing instruction to buffer the response is that the Web Publishing Engine can determine the length of the response and set the `Content-Length` header in the response. Buffering the response might reduce the Web Publishing Engine’s performance.

Using Web Publishing Engine sessions to store information between requests

You can use the Web Publishing Engine's server-side sessions to track and store any type of information between requests. Sessions allow you to create a web application that is able to maintain state by using persistent arbitrary pieces of information between requests. For example, user client information that is entered on a first form page could be stored in a session and then used to populate values on a subsequent page.

By default, the Web Publishing Engine will use a cookie to store the session ID. To accommodate clients that do not allow cookies, you can use the `fmxml:session_encode_url()` function to add the Session ID to the URL. To guarantee compatibility in all situations, it is recommended that you encode all URLs written out to the page with the `fmxml:session_encode_url()` function. This function adds to your URL a semicolon-separated parameter called `jsessionid`, which is the identifier for the particular client's parent session.

For example, instead of placing the following link on a page:

```
<a href="my_stylesheet.xml?-db=products&-lay=sales&-grammar=fmresultset&-findall">hyperlinked text</a>
```

You should encode all links on a page as follows:

```
<a href="{fmxml:session_encode_url('my_stylesheet.xml?-db=products&-lay=sales&-grammar=fmresultset
&-findall')}">hyperlinked text</a>
```

If the client does not allow cookies, the page includes:

```
<a href="my_stylesheet.xml;jsessionid=<session id>?-db=products&-lay=sales&-grammar=fmresultset&-findall">
hyperlinked text</a>
```

If the Web Publishing Engine detects that the client allows cookies, then the `fmxml:session_encode_url()` function stores the session ID in a cookie instead of the URL.

Note Session information does not persist after the Web Publishing Engine is restarted.

Using the session extension functions

Use the following session extension functions to manipulate session variables. You can store a string, number, boolean value, or `node-set` in a session object. By using `node-set`, you can create a data structure in XML and then store it between requests in the session object.

Session extension function	Data type returned	Description
<code>fmxml:session_exists(String session-name)</code>	boolean	Checks if a session with the specified name exists.
<code>fmxml:create_session(String session-name)</code>	boolean	Creates a session with the specified session name and the default time-out, which is set via the Admin Console. See FileMaker Server Help. Note This function requires the <code><?xslt-cwp-buffer?></code> processing instruction. See "Using content buffering" on page 63.
<code>fmxml:invalidate_session(String session-name)</code>	boolean	Forces the session to time out immediately.
<code>fmxml:set_session_timeout(String session-name, Number timeout)</code>	boolean	Sets the session timeout in seconds. The default timeout for sessions is set via the Admin Console.
<code>fmxml:session_encode_url(String url)</code>	string	Encodes a URL with the session ID if the client does not support cookies; otherwise returns input URL.

Session extension function	Data type returned	Description
fmxls:set_session_object(String session-name, String name, Object value)	XSLT object (number, string, boolean, or node-set)	Stores an XSLT object (a number, string, boolean, or node-set) under a session, which can be later retrieved using the fmxls:get_session_object () function. This function also returns the previously stored object under the specified session object name. If nothing was stored under the name, it returns a null object. Note The set_session_object() extension function stores string values only; it interprets any object passed to it as a string. If the object cannot be converted to a string, then no value is stored in the session and the extension function error code is set to 10100 (Unknown Session Error). If you attempt to set a session object using null or an empty string, you also receive an error code of 10100 (Unknown Session Error). To clear out a session variable, remove the variable from the session using the remove_session_object() function.
fmxls:get_session_object(String session-name, String name)	XSLT object	Retrieves an XSLT object from the session.
fmxls:remove_session_object(String session-name, String name)	XSLT object	Returns and then removes an XSLT object from the session.

Here is an example of creating a session and then storing a favorite color in the session:

```
<xsl:variable name="session">
  <xsl:choose>
    <xsl:when test="not (fmxls:session_exists(string($session-name)))">
      <xsl:value-of select="fmxls:create_session(string($session-name))"/>
    </xsl:when>
    <xsl:otherwise>true</xsl:otherwise>
  </xsl:choose>
</xsl:variable>
<xsl:variable name="favorite-color" select="fmxls:set_session_object(string($session-name), 'favorite-color', string($color))"/>
```

Important

- To make sure that users are logged out of a database after completing a session, use the fmxls:invalidate_session () function to force the session to time out immediately.
- If you are using global fields or a script that sets or modifies a state, you must use the Admin Console to enable the XSLT Database Sessions option for the Web Publishing Engine. Otherwise, global field values and states are not maintained between requests. See FileMaker Server Help.
- If you switch to another database file when using Web Publishing Engine sessions, global field values are not preserved. The Web Publishing Engine closes the first file before opening the second file. As an alternative, you can access data from the second database file by using a layout in the first database file.

Sending email messages from the Web Publishing Engine

You can use the Web Publishing Engine to generate email messages, which is useful for custom web solutions. To have the Web Publishing Engine send an email message, use one of the following three `fmxls:send_email()` extension functions in an XSLT stylesheet. You can use these functions to send one or more separate messages. Because the `fmxls:send_email()` functions are contained in the Web Publishing Engine's server-side XSLT stylesheet, a client cannot use the Web Publishing Engine to send unauthorized email messages.

Email extension function	Data type returned	Description
<code>fmxls:send_email(String smtpFields, String body)</code>	boolean	Sends a plain text email message of any length from the Web Publishing Engine using the Web Publishing Engine's default text encoding for email messages
<code>fmxls:send_email(String smtpFields, String body, String encoding)</code>	boolean	Sends a plain text email message of any length using one of the following text encodings: US-ASCII, ISO-8859-1, ISO-8859-15, ISO-2022-JP, Shift_JIS, UTF-8. For information on these encodings, see "Setting text encoding for requests" on page 57.
<code>fmxls:send_email(String smtpFields, String xsltFile, Node xml, boolean includeImages)</code>	boolean	Sends an HTML-based email message using the encoding that is specified by encoding attribute of the <code><xsl:output></code> element in the stylesheet. If the encoding attribute is not included in the <code><xsl:output></code> element, the Web Publishing Engine's default text encoding for email messages is used.

Notes

- In each of the three forms of the `fmxls:send_email()` function, the `smtpFields` parameter is a URL-encoded string of any length that contains the address and subject information using the following format, which is based on RFC 2368, the `mailto` URL scheme:

`username@host?name1=value1&name2=value2...`

where `username@host` specifies a recipient. The name/value pairs can be specified in any order and are defined as follows:

- `from=username@host` (must appear only once). The `from` field must be specified.
- `to=username@host`. Use this name/value pair for additional recipients.
- `reply-to=username@host` (can appear only once)
- `cc=username@host`
- `bcc=username@host`
- `subject=string` (can appear only once)

If the `from`, `reply-to`, or `subject` fields are specified more than once, then the email message is not sent, a value of `false()` is returned by the function, and the appropriate error status code is set.

- The Web Publishing Engine will check the syntax of all email addresses provided. They must be of the form: `user@host.tld` or "quoted identifier"<`user@host.tld`> where `tld` is any top-level-domain such as `com` or `net`. If any of the fields contains an invalid email address, then the email message is not sent and the appropriate error status code is set.
- The individual values for the `smtpFields` parameter, such as the subject, must be a URL-encoded string. For example, the "&" character must be specified as "&" and blank spaces must be specified as "%20". The entire string for the `smtpFields` parameter must be XML-encoded. (See the example at the end of this section.)

- For each of these functions, a value of `true()` is returned if the email message is successfully sent; otherwise `false()` is returned.
- For English email messages, the Web Publishing Engine uses an initial default text encoding of ISO-8859-1. You can change this setting by using the Admin Console. See FileMaker Server Help.
- The `fmxls:send_email(String smtpFields, String xsltFile, Node xml, boolean includelImages)` function sends an email message consisting of XML data that is processed by the email stylesheet you specify in this function.
 - For the `xsltFile` parameter, specify the name of the email stylesheet by entering a URL that is relative to the main processing stylesheet file for the request.
 - For the `xml` parameter, specify the parent node of the XML data that you want to use with the email stylesheet. To send an email message using the same XML data that is being displayed in the browser, simply provide the XPath for the root of the document: `"/`. Otherwise, you can use a different XML document by first loading it with the `document()` function, and then passing that document into the `fmxls:send_email()` function.
 - For the `includelImages` parameter, specify a boolean value of `true()` to have the Web Publishing Engine include all images specified in the HTML of the email message as attachments. This parameter includes both FileMaker database images as well as non-database images from other locations. The Web Publishing Engine changes the image URLs to refer to the attachments. Performance can be slow if the image files are numerous or large. If you specify `false ()`, the Web Publishing Engine does not change the URLs for the images. If the URLs are absolute, the email client will attempt to load the images from the web server.

Here is an example of using the `fmxls:send_email(String smtpFields, String xsltFile, Node xml, boolean includelImages)` function inside an XPath statement, such as inside the `<xsl:if>` element:

```
fmxls:send_email('tom_jones@company.com?subject=project%20status&from=john_smith@company.com
&cc=jane_doe@company.com','my_mail_template.xml',/, true())
```

For information about configuring the Web Publishing Engine to connect to a SMTP server, see FileMaker Server Help.

Using the header functions

You can use the `fmxls:get_header()` function to read information from the HTTP request and response headers, and the `fmxls:set_header()` function to write information to the headers. These functions are useful if the client can use the header information to retrieve information from the web server, or if you need to set a HTTP header for other reasons.

Header extension function	Data type returned	Description
<code>fmxls:get_header(String name)</code>	string	Returns the specified header value
<code>fmxls:set_header(String name, String value)</code>	void	Sets the specified header value
<code>fmxls:set_status_code(Number status-code)</code>	void	Sets the HTTP status code

Notes

- The name used in the `fmxslt:get_header()` and `fmxslt:set_header()` functions, and the value in the `fmxslt:set_header()` function can be a string of any length.
- The `fmxslt:set_header()` function and the `fmxslt:set_status_code()` function require the `<?xslt-cwp-buffer?>` processing instruction. See “Using content buffering” on page 63.

The following example demonstrates how to set the value of the header. Suppose you are using a stylesheet to output a vCard. There is a potential problem that when a browser tries to load the stylesheet page, the browser could interpret the `.xsl` file as a stylesheet rather than a vCard. If you use the header called Content-Disposition, you can specify that there is an attachment with an extension of `.vcf`.

```
<xsl:value-of select="fmxslt:set_header('Content-Disposition','attachment;filename=test.vcf')"/>
```

Using the cookie extension functions

You can use the cookie extension functions to get or set cookies stored in the client’s web browser.

Cookie extension function	Data type returned	Description
<code>fmxslt:get_cookie(String name)</code>	node-set	Returns the COOKIES node-set that has the specified cookie name.
<code>fmxslt:get_cookies()</code>	node-set	Returns COOKIES node-set with all of the cookies supplied by the client.
<code>fmxslt:set_cookie(String name, String value)</code>	void	Stores the specified cookie in the client’s browser with the specified value.
<code>fmxslt:set_cookie(String name, String value, Number expires, String path, String domain)</code>	void	Stores the specified cookie in the client’s browser with all of the values available for a cookie. The Expires parameter is the number of seconds until the cookie expires.

Notes

- The `fmxslt:get_cookie()` and `fmxslt:get_cookies()` functions return a node-set in the following structure:


```
<!ELEMENT cookies (cookie)*>
  <!ATTLIST cookie xmlns CDATA #FIXED "http://www.filemaker.com/xml/cookie">
<!ELEMENT cookie (#PCDATA)>
  <!ATTLIST cookie name CDATA #REQUIRED>
```
- The XML namespace for the cookies node-set is "http://www.filemaker.com/xml/cookie". You must declare the namespace and provide a prefix for the namespace.
- All of the parameter values for the `fmxslt:set_cookie` functions must be valid or else the web browser will ignore the `fmxslt:set_cookie` function requests.
- For all cookie functions, the string parameters can be any length.
- Both forms of the `fmxslt:set_cookie()` function require the `<?xslt-cwp-buffer?>` processing instruction. See “Using content buffering” on page 63.

Example: get_cookie

The following example retrieves a cookie named preferences and its value:

```
<xsl:variable name="pref_cookie" select="fmxslt:get_cookie('preferences')"/>
<xsl:value-of select="concat('Cookie Name = ', $pref_cookie/fmc:cookies/fmc:cookie/@name)"/> <br/>
<xsl:value-of select="concat('Cookie Value = ', $pref_cookie/fmc:cookies/fmc:cookie)"/>
```

Example: set_cookie

Here is an example of how to set a cookie with all values:

```
<xsl:variable name="storing_cookie" select="fmxslt:set_cookie ('text1', 'text2', 1800, 'my_text', 'my.company.com')"/>
```

Using the string manipulation extension functions

You can use the string manipulation functions to change the encoding of strings of any length.

String manipulation extension function	Data type returned	Description
fmxslt:break_encode(String value)	string	Returns an HTML break-encoded string. Characters such as “&” (ampersand) are replaced with “&”. New line characters such as line feeds and carriage returns are replaced with . This function works only if the disable-output-escaping attribute of the <xsl:value-of> and <xsl:text> elements is set to “yes” (disable-output-escaping=“yes”). Note To include a line feed or carriage return in the string that the fmxslt:break_encode() function is applied to, you must use the following escape characters in the string: “
” (for line feed) or “” (for carriage return). You cannot include a line feed or carriage return in the string by pressing the return key in your text editor.
fmxslt:html_encode(String value)	string	Returns an HTML-encoded string; characters such as “&” (ampersand) are replaced with “&”
fmxslt:url_encode(String value)	string	Returns a URL-encoded string. URL encoding is used to transmit characters over the Internet, particularly for URLs. For example, the “&” (ampersand) in a URL-encoded form is %26. If a reserved character is used in your href, use this function to URL-encode your string.
fmxslt:url_encode(String value, String encoding)	string	Returns a URL-encoded string using the character encoding you specify for the encoding parameter, which can be: US-ASCII, ISO-8859-1, ISO-8859-15, ISO-2022-JP, Shift_JIS or UTF-8. Use this function in situations where you know that a web server will be expecting a different character encoding than the one used in your current stylesheet. For example, your website entrance page might be displayed in UTF-8, but users may click a link to go to a Japanese page. If the request includes Japanese characters and the Japanese pages use Shift_JIS encoding, it is best to encode the string in Shift_JIS.

String manipulation extension function	Data type returned	Description
<code>fmxs:url_decode(String value)</code>	string	Returns a URL-decoded string from a URL string that was previously encoded.
<code>fmxs:url_decode(String value, String encoding)</code>	string	Returns a URL-decoded string using the character encoding you specify for the encoding parameter, which can be: US-ASCII, ISO-8859-1, ISO-8859-15, ISO-2022-JP, Shift_JIS or UTF-8. Use this function in situations where you must specify the character encoding used in a URL encoded string in order to decode the string properly. For example, even though your website uses ISO-8859-1, users might submit a form using a different character encoding.

Comparing strings using Perl 5 regular expressions

You can use the `fmxs:regex_contains()` extension function to compare strings using Perl 5 regular expressions. A regular expression comparison is an advanced type of text matching that enables you to determine if a string matches a specified pattern. The syntax of this function is:

```
fmxs:regex_contains(String input, String pattern)
```

where `input` is a string and `pattern` is a Perl 5 regular expression. For more information on the syntax of Perl 5 regular expressions, see www.perldoc.com. The `fmxs:regex_contains()` function returns a boolean value.

This function is useful if you need more advanced string manipulation than is provided by standard XSLT. For example, you can determine if a field value contains a valid telephone number or email address by comparing the string against a Perl 5 regular expression.

Here is an example of using this function to determine if a field value contains email addresses that are constructed correctly:

```
<xsl:variable name="email" select="'foo@bar.com'"/>
<xsl:if test="fmxs:regex_contains($email, '^\\w+[\\w-\\.]*@\\w+((-\\w+)|(\\w*))\\. [a-z]{2,3}$')">Valid Email</xsl:if>
```

If the Web Publishing Engine cannot parse the pattern, the error status is set to error code 10311. See “Error code numbers for the FileMaker XSLT extension functions” on page 115.

Checking for values in a field formatted as a checkbox

You can use the following extension function to determine whether a particular value in a checkbox value list is stored in a field in the FileMaker database:

```
fmxs:contains_checkbox_value(String valueString, String valueListEntry)
```

where `valueString` is an XPath specifying the field, and `valueListEntry` is the value you want to check for.

If the specified value is stored in the field, this boolean function returns `true()`. Otherwise, it returns `false()`. You can use this function to determine whether to set the checked attribute in an HTML form to display a checkbox as being selected.

For example, suppose a field in a FileMaker database layout has the following checkbox options:

- Red
- Blue
- Green
- Small
- Medium
- Large

If a user selected Red only, then the field would contain the string “Red”. To determine whether the field contains “Blue”, you could use the following function call:

```
fmxslt:contains_checkbox_value(<field value node>,'Blue')
```

where <field value node> is the XPath to the <data> element for the checkbox field. The function would return “false” in this example.

A common application of this function is to display the checkbox value list in a web page and select the checkboxes in the web page that are selected in the database. For example, the following HTML and XSLT statements create a set of checkboxes for a field named style using a value list named color_size:

```
<xsl:variable name="field-value" select="fmrs:field[@name='style']/fmrs:data" />
<xsl:for-each select="$valueLists[@NAME = 'color_size']/fml:VALUE">
  <input type="checkbox">
    <xsl:attribute name="name">style</xsl:attribute>
    <xsl:attribute name="value"><xsl:value-of select="."/></xsl:attribute>
    <xsl:if test="fmxslt:contains_checkbox_value($field-value,.)">
      <xsl:attribute name="checked">checked</xsl:attribute>
    </xsl:if>
  </input><xsl:value-of select="."/><br/>
</xsl:for-each>
```

The HTML and XSLT statements in the example would output the following checkboxes on a web page, with Red and Medium selected:

- Red
- Blue
- Green
- Small
- Medium
- Large

Using the date, time, and day extension functions

You can use extension functions to get the current date, time, or day, and to compare any two dates, times, or days.

The functions in the following table use the “fm” formats regardless of locale. The “fm” formats are MM/dd/yyyy for date, HH:mm:ss for time, and MM/dd/yyyy HH:mm:ss for timestamp.

To re-arrange output values into a different or preferred format, use calculation functions or JavaScript.

Date, time, day extension function	Data type returned	Description
fmxls: get_date()	string	Returns the current date in the “fm” format.
fmxls: get_date(String format)	string	Returns the current date in the format you specify. Enter the string “short”, “long”, or “fm” for the format parameter.
fmxls: get_time()	string	Returns the current time in the “fm” format.
fmxls: get_time(String format)	string	Returns the current time in the format you specify. Enter the string “short”, “long”, or “fm” for the format parameter.
fmxls: get_day()	string	Returns the current day in the short format.
fmxls: get_day(String format)	string	Returns the current day in the format you specify. Enter the string “short” or “long” for the format parameter.
fmxls: get_fm_date_format()	string	Returns the formatting string for “fm” date format: “MM/dd/yyyy”
fmxls: get_short_date_format()	string	Returns the formatting string for short date format: “M/d/yy”
fmxls: get_long_date_format()	string	Returns the formatting string for long date format: “MMM d, yyyy”
fmxls: get_fm_time_format()	string	Returns the formatting string for “fm” time format: “HH:mm:ss”
fmxls: get_fm_timestamp_format()	string	Returns the formatting string for “fm” timestamp format: “MM/dd/yyyy HH:mm:ss”
fmxls: get_short_time_format()	string	Returns the formatting string for short time format: “h:mm a”
fmxls: get_long_time_format()	string	Returns the formatting string for long time format: “h:mm:ss a z”
fmxls: get_short_day_format()	string	Returns the formatting string for short day format: “EEE”
fmxls: get_long_day_format()	string	Returns the formatting string for long day format: “EEEE”
fmxls: compare_date(String date1, String date2)	number	This function compares two date values. It returns a negative number if date1 is before date2. It returns a positive number if date1 is after date2. It returns a 0 if date1 is identical to date2. Both dates must be specified in the “fm” date format.

Date, time, day extension function	Data type returned	Description
<code>fmxls:compare_time(String time1, String time2)</code>	number	This function compares two time values. It returns a negative number if <code>time1</code> is before <code>time2</code> . It returns a positive number if <code>time1</code> is after <code>time2</code> . It returns a 0 if <code>time1</code> is identical to <code>time2</code> . Both times must be specified in the “fm” time format.
<code>fmxls:compare_day(String day1, String day2)</code>	number	This function compares two day values. It returns a negative number if <code>day1</code> is before <code>day2</code> . It returns a positive number if <code>day1</code> is after <code>day2</code> . It returns a 0 if <code>day1</code> is identical to <code>day2</code> . Both days must be specified in the short day format.

The functions in the following table use custom date formatting strings that specify a date and time format. See the next section, “About the date and time format strings.”

Date, time, day extension function	Data type returned	Description
<code>fmxls:get_datetime(String dateFormat)</code>	string	Returns the current date and time using the date and time format strings.
<code>fmxls:convert_datetime(String oldFormat, String newFormat, String date)</code>	string	Converts the specified date in the specified <code>oldFormat</code> into the string according to the specified <code>newFormat</code> . The <code>oldFormat</code> and <code>newFormat</code> strings must be specified using the date and time format strings.
<code>fmxls:compare_datetime(String dateFormat1, String dateFormat2, String date1, String date2)</code>	number	This function compares <code>date1</code> and <code>date2</code> by decoding these dates according to respective date formats. It returns a negative number if <code>date1</code> is before <code>date2</code> . It returns a positive number if <code>date1</code> is after <code>date2</code> . It returns a 0 if <code>date1</code> is identical to <code>date2</code> . The <code>dateFormat1</code> and <code>dateFormat2</code> strings must be specified using the date and time format strings.

About the date and time format strings

The date and time formats are specified by date and time pattern strings. Within date and time pattern strings, unquoted letters from ‘A’ to ‘Z’ and from ‘a’ to ‘z’ are interpreted as pattern letters representing the components of a date or time string.

The following pattern letters are defined (all other characters from ‘A’ to ‘Z’ and from ‘a’ to ‘z’ are reserved):

Letter	Date or Time Component	Presentation	Examples
G	Era designator	Text	AD
y	Year	Year	1996; 96
M	Month in year	Month	July; Jul; 07
w	Week in year	Number	27
W	Week in month	Number	2
D	Day in year	Number	189
d	Day in month	Number	10
F	Day of week in month	Number	2
E	Day in week	Text	Tuesday; Tue

Letter	Date or Time Component	Presentation	Examples
a	Am/pm marker	Text	PM
H	Hour in day (0-23)	Number	0
k	Hour in day (1-24)	Number	24
K	Hour in am/pm (0-11)	Number	0
h	Hour in am/pm (1-12)	Number	12
m	Minute in hour	Number	30
s	Second in minute	Number	55
S	Millisecond	Number	978
z	Time zone	General time zone	Pacific Standard Time; PST; GMT-08:00
Z	Time zone	RFC 822 time zone	-0800

Pattern letters are usually repeated, as their number determines the exact presentation:

- **Text:** For formatting, if the number of pattern letters is four or more, the full form is used; otherwise a short or abbreviated form is used if available. For parsing, both forms are accepted, independent of the number of pattern letters.
- **Number:** For formatting, the number of pattern letters is the minimum number of digits, and shorter numbers are zero-padded to this amount. For parsing, the number of pattern letters is ignored unless it's needed to separate two adjacent fields.
- **Year:** For formatting, if the number of pattern letters is two, the year is truncated to two digits; otherwise it is interpreted as a number.

For parsing, if the number of pattern letters is more than 2, the year is interpreted literally, regardless of the number of digits. So using the pattern “MM/dd/yyyy”, “01/11/12” parses to Jan 11, 12 A.D.

For parsing with the abbreviated year pattern (“y” or “yy”), the abbreviated year must be interpreted relative to some century by adjusting dates to be within 80 years before and 20 years after the time the date format instance is created. For example, using a pattern of “MM/dd/yy” and a date format instance created on Jan 1, 1997, the string “01/11/12” would be interpreted as Jan 11, 2012 while the string “05/04/64” would be interpreted as May 4, 1964. During parsing, only strings consisting of exactly two digits will be parsed into the default century. Any other numeric string, such as a one digit string, a three or more digit string, or a two digit string that isn't all digits (for example, “-1”), is interpreted literally. So “01/02/3” or “01/02/003” are parsed, using the same pattern, as Jan 2, 3 AD. Likewise, “01/02/-3” is parsed as Jan 2, 4 BC.

- **Month:** If the number of pattern letters is three or more, the month is interpreted as text; otherwise, it is interpreted as a number.

- **General time zone:** Time zones are interpreted as text if they have names. For time zones representing a GMT offset value, the following syntax is used:
 - GMTOffsetTimeZone. *GMT Sign Hours:Minutes*
 - Sign. + or -
 - Hours. *Digit* or *Digit Digit*
 - Minutes. *Digit Digit*
 - Digit. One of the following: 0 1 2 3 4 5 6 7 8 9

Hours must be between 0 and 23, and *Minutes* must be between 00 and 59. The format is locale independent and digits must be taken from the Basic Latin block of the Unicode standard.

For parsing, RFC 822 time zones are also accepted.

- **RFC 822 time zone:** For formatting, the RFC 822 4-digit time zone format is used:
 - RFC822TimeZone. *Sign TwoDigitHours Minutes*
 - TwoDigitHours. *Digit Digit*

TwoDigitHours must be between 00 and 23. Other definitions are as for general time zones.

For parsing, general time zones are also accepted.

The following examples show how date and time patterns are interpreted in the U.S. locale. The given date and time are 2001-07-04 12:08:56 local time in the U.S. Pacific Time time zone.

Date and Time Pattern	Result
"yyyy.MM.dd G 'at' HH:mm:ss z"	2001.07.04 AD at 12:08:56 PDT
"EEE, MMM d, ' 'yy"	Wed, Jul 4, '01
"h:mm a"	12:08 PM
"hh 'o' 'clock' a, zzzz"	12 o'clock PM, Pacific Daylight Time
"K:mm a, z"	0:08 PM, PDT
"yyyyy.MMMMM.dd GGG hh:mm aaa"	02001.July.04 AD 12:08 PM
"EEE, d MMM yyyy HH:mm:ss Z"	Wed, 4 Jul 2001 12:08:56 -0700
"yyMMddHHmmssZ"	010704120856-0700

Copyright 2003 Sun Microsystems, Inc. Reprinted with permission.

Checking the error status of extension functions

You can use the following extension function within an XSLT stylesheet to check the current error status of the most recently called FileMaker XSLT extension function and handle errors that occur during the processing of your pages:

```
fmxslt:check_error_status()
```

When the `fmxslt:check_error_status()` function is called, the Web Publishing Engine returns the current error code value for the most recently called function as a Number type, and then resets the error status to 0 (“No Error”). For information on the error code values, see “Error code numbers for the FileMaker XSLT extension functions” on page 115.

Using logging

You can use the standard XSLT `<xsl:message>` element to write log entries to the Web Publishing Engine application log file. See “Using the Web Publishing Engine application log” on page 84.

Using server-side processing of scripting languages

The underlying XSLT transformer embedded in the Web Publishing Engine supports server-side processing of scripting languages. As a result, you can use JavaScript to develop your own extension functions that can be called directly from an XSLT stylesheet.

Two Java libraries are installed to enable this functionality:

- `bsf.jar` – This library allows the XSLT transformer to connect to scripting languages.
- `js.jsr` – This library is a full JavaScript implementation from the Mozilla project.

With these libraries, you can create your own extension functions inside your XSLT stylesheet code. These extension functions can implement any scripting logic and can be more manageable than relying on XSLT and XPath to accomplish logical functions.

You can find more detailed information about the extension support of the XSLT transformer on the Apache Xalan Extensions website:

<http://xml.apache.org/xalan-j/extensions.html>

Defining an extension function

To define an extension function inside your stylesheet:

1. Define the namespace for the extension.

Add the `xalan` namespace to instruct the XSLT transformer to support extension components, providing the name for your own extension function namespace. The following example uses `fmp-ex` as the extension function namespace prefix.

```
<xsl:stylesheet version="1.0"
xmlns:xsl=http://www.w3.org/1999/XSL/Transform
xmlns:xalan=http://xml.apache.org/xslt
xmlns:fmp-ex="ext1"
exclude-result-prefixes="xsl xalan fmp-ex">
```

2. Define the extension component and extension functions, with the code that actually implements your extension function.

```
<xalan:component prefix="fmp-ex" functions="getValueColor">
<xalan:script lang="javascript">
  function getValueColor(value) {
    if (value > 0)
      return ("#009900");
    else
      return ("#CC0000");
  }
</xalan:script>
</xalan:component>
```

This example returns a color value based on an input value. If the input value is greater than 0, the color returned is green ("#009900"); otherwise, if the value is less than 0, the color returned is red ("#CC0000").

Note The `<xalan:component>` element needs to be the child of `<xsl:stylesheet>` element.

3. Use the extension function inside the stylesheet.

The following examples show how to call an extension function using an XPath statement.

This first example would set the font color to green ("#009900").

```
<font color="{fmp-ex:getValueColor(50)}">The value is 50</font>
```

This second example would set the font color to red ("#CC0000").

```
<font color="{fmp-ex:getValueColor(-500)}">The value is -500</font>
```

An extension function example

The simple JavaScript function used in the process above could have been implemented using an `<xsl:choose>` statement. But the real power of using a scripting extension is that you can create a function that cannot be implemented in XSLT or XPath alone.

For example, say that you are building an intranet portal site for your company, and you want to include the current stock price information on that portal page. While there are XML stock feeds available, generally they require commercial licenses to access them. However, you can download stock data in a Comma Separated Values (CSV) document format from the Yahoo! website. The XPath `document()` function can import content from XML sources, but you would need to convert the CSV content into XML. One solution is to use JavaScript to download the CSV stock price information, parse the file, and extract the data.

This URL shows the syntax for retrieving a stock quote from the Yahoo! website as a CSV file:

```
http://quote.yahoo.com/d/quotes.csv?s=<ticker>&f=l1gh&e=.csv
```

where `<ticker>` represents the stock symbol that you are trying to retrieve data for.

The data returned is three numbers separated by commas, such as:

```
31.79,31.17,32.12
```

where the first value is the last trade price, the second value is the daily low, and the third value is the daily high.

The example below shows a JavaScript XSLT extension function that retrieves a current stock price from the Yahoo! website and makes it available via an XPath function:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  exclude-result-prefixes="xsl fmxslt fmrs xalan fmp-ex"
  version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fmrs="http://www.filemaker.com/xml/fmresultset"
  xmlns:fmxslt="xalan://com.fmi.xslt.ExtensionFunctions"
  xmlns:xalan="http://xml.apache.org/xslt"
  xmlns:fmp-ex="ext1"
>

<?xslt-cwp-query params="-grammar=fmresultset&-process" ?>
<xsl:output method="html"/>
<xalan:component prefix="fmp-ex" functions="include get_quote" >
<xalan:script lang="javascript">
  function include(url) {
    var dest = new java.net.URL(url);
    var dis = new java.io.DataInputStream(dest.openStream());
    var res = "";
    while ((line = dis.readLine()) != null)
    {
      res += line + java.lang.System.getProperty("line.separator");
    }
    dis.close();
    return res;
  }
  function get_quote(ticker) {
    url = "http://quote.yahoo.com/d/quotes.csv?s="+
      "+ticker+"&f=l1gh&e=.csv";
    csv_file = include(url);
    var str_tokenizer = new java.util.StringTokenizer(csv_file, ',');
    // the first token is the last trade price
    var last = str_tokenizer.nextToken();
    return last;
  }
</xalan:script>
</xalan:component>
```

```
<xsl:template match="/fmrs:fmresultset">
  <html>
    <body>
      <font size="2" face="verdana, arial">
        Apple Stock Price: <xsl:value-of select="fmp-ex:get_quote('AAPL')"/>
      </font>
    </body>
  </html>
</xsl:template>

</xsl:stylesheet>
```

When the Web Publishing Engine processes this stylesheet, it requests the stock information from the Yahoo! website. The `get_quote()` function parses the stock quote data and returns the data to the stylesheet. The transformed output appears in the browser.

Chapter 7

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).

To stage your site and prepare it for testing:

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

Note For instructions, see the FileMaker Server Help.

3. Verify that the web server and the Web Publishing Engine are running.
4. Copy or move the XSLT stylesheets to the machine on which the Web Publishing Engine resides.

Copy or move the XSLT stylesheets to the following directory on the Web Publishing Engine machine:

- Apache (Mac OS): /Library/FileMaker Server/Web Publishing/xslt-template-files
- IIS (Windows): <drive>\Program Files\FileMaker\FileMaker Server\Web Publishing\xslt-template-files

where <drive> is the primary drive from which the system is started.

Note You can also place the stylesheets in an optional folder or folder hierarchy in the xslt-template-files folder.

5. Copy or move any referenced container objects to the web server machine.

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.

Note 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.

6. Use the following URL syntax to request and process an XSLT stylesheet, and generate the resulting HTML:

```
<scheme>://<host>[:<port>]/fmi/xsl/<path>/<stylesheet>.xsl[?<query string>]
```

where:

- <scheme> is the HTTP or HTTPS protocol.
- <host> is the IP address or domain name of the host computer where the web server is installed.
- <port> is optional and specifies the port that the web server is listening on. If no port is specified, then the default port for the protocol is assumed (port 80 for HTTP, or port 443 for HTTPS).
- <path> is optional and specifies the folder(s) inside the xslt-template-files folder where the XSLT stylesheet is located.
- <stylesheet> is the name of the stylesheet with an .xsl extension.
- <query string> can be a combination of one query command and one or more query parameters for Custom Web Publishing with XSLT.

If the specified stylesheet includes a `<?xslt-cwp-query?>` processing instruction, the statically-defined query command and parameters take precedence over any matching query command or parameters in the URL query string. If you are referencing a `home.xml` stylesheet generated by XSLT Site Assistant, you need not include a query string. See appendix A, “Valid names used in query strings” for more information on query strings.

The URL syntax, including the names of query commands and parameters, is case sensitive except for portions of the query string. The majority of the URL is in lower case. For example, if you copied your stylesheets (including a `home.xml` stylesheet) into the `my_templates` folder inside the `xslt-template-files` folder, you can use the following URL to request and process the stylesheets:

```
http://192.168.123.101/fmi/xsl/my_templates/home.xml
```

Note The Web Publishing Engine does not allow web users to view the source for XSLT stylesheets that are installed in the `xslt-template-files` folder. When a web user sends a request to process a stylesheet, the Web Publishing Engine sends to the web browser only the HTML pages that are the result of XSLT Site Assistant's stylesheets.

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 22 for information on designing web-friendly scripts.
- Test your site with different operating systems and web browsers.

Note If you don't have a network connection and you have installed FileMaker Server using a single machine deployment—with the web server, Web Publishing Engine, and Database Server on one computer—then you can test your Custom Web Publishing site by using `http://localhost/` or `http://127.0.0.1/` in the URL. For information on the URL syntax, see “About the URL syntax for XML data and container objects” on page 35, and “About the URL syntax for FileMaker XSLT stylesheets” on page 52.

Examples of stylesheets for testing XML output

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. On the same page, you can compare the transformed XSLT results against the XML source document before the transformation.

```
<?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 application log
- Web Server Module error log
- Web Publishing Core internal access logs

Using the web server access and error logs

Apache (Mac OS): The Apache web server generates an access log file and an error log file. The Apache access log file, which is in the W3C Common Logfile Format by default, is a record of all incoming HTTP requests to the web server. The Apache error log is a record of problems involving processing HTTP requests. For more information on these log files, see the documentation for the Apache web server.

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

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 application log

By default, the Web Publishing Engine generates an application log file that is a record of Web Publishing Engine error, script, and user log information.

- The error log information describes any unusual Web Publishing Engine errors that have occurred. Common errors reported to the web user, such as “Database not open,” are not recorded.
- The script log information describes any errors generated when web users execute scripts. For example, it lists script steps that are skipped if they’re not web-compatible.
- The user log messages contains messages generated by the XSLT `<xsl:message>` element in XSLT stylesheets. Whenever web users access your XSLT stylesheet, information you’ve included within a `<xsl:message>` element is recorded in the application log file. See chapter 6, “Developing FileMaker XSLT stylesheets.”

The application log is named `pe_application_log.txt` and is located in the Logs folder in the FileMaker Server folder on the Web Publishing Engine host.

The `pe_application_log.txt` file is generated if any of the following Logging options are enabled in the Web Publishing Engine.

Logging option enabled	Information recorded in <code>pe_application_log.txt</code>
Error Logging	Any unusual Web Publishing Engine errors that have occurred. Common errors reported to the web user, such as “Database not open,” are not recorded.
Script Logging	Any errors generated when web users execute scripts. For example, it lists script steps that are skipped if they’re not web-compatible.
User Logging	Messages generated whenever web users access your Custom Web Publishing solution.

All three of these Logging options are enabled by default. For information on setting these options via the Admin Console, see FileMaker Server Help.

Note The entries in the application log are not automatically deleted, and over time the file may become very large. To save hard disk space on your host computer, archive the application log file on a regular basis.

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 Web Publishing Core internal access logs

The Web Publishing Core software component of the Web Publishing Engine generates two internal access log files that are a record of each time the Web Publishing Core is accessed.

- The `wpc_access_log.txt` access log is a record of all end-user requests to generate XML and to use FileMaker Server Instant Web Publishing. These requests are routed from the web server directly to the Web Publishing Core.
- The `pe_internal_access_log.txt` access log is a record of all internal XML requests that the XSLT-CWP software component of the Web Publishing Engine makes while processing XSLT requests. These requests are routed internally within the Web Publishing Engine from the XSLT-CWP software component to the Web Publishing Core software component.

These log files are located in the `Logs` folder in the `FileMaker Server` folder on the Web Publishing Engine host.

The internal access logs are generated if the `Access Logging` option is enabled in the Web Publishing Engine. The default setting is enabled. For information on setting the `Access Logging` option in the Admin Console, see `FileMaker Server Help`.

Appendix A

Valid names used in query strings

This appendix describes the valid names of query commands and parameters you can use in a 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

- dbnames (See page 91.)
- delete (See page 91.)
- dup (See page 92.)
- edit (See page 92.)
- find, -findall, -findany (See page 92.)
- findquery (See page 93.)
- layoutnames (See page 93.)
- new (See page 93.)
- process (XSLT only) (See page 94.)
- scriptnames (See page 94.)
- view (See page 94.)

Query parameter names

- db (See page 95.)
- encoding (XSLT only) (See page 95.)
- field (See page 95.)
- fieldname (See page 96.)
- fieldname.op (See page 97.)
- grammar (XSLT only) (See page 98.)
- lay (See page 98.)
- lay.response (See page 98.)
- lop (See page 98.)
- max (See page 99.)
- modid (See page 99.)
- query (See page 99.)
- recid (See page 100.)
- relatedsets.filter (See page 100.)
- relatedsets.max (See page 101.)
- script (See page 101.)
- script.param (See page 101.)
- script.prefind (See page 102.)
- script.prefind.param (See page 102.)
- script.presort (See page 102.)
- script.presort.param (See page 103.)
- skip (See page 103.)
- sortfield.[1-9] (See page 104.)
- sortorder.[1-9] (See page 104.)
- stylehref (See page 105.)
- styletype (See page 105.)
- token.[string] (XSLT only) (See page 105.)

Important The -lay parameter for specifying a layout is required with all query commands except -dbnames, -layoutnames, -scriptnames, and -process (XSLT requests only).

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` and `–process` 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 46.
- 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`.
- Query command and parameter names must be specified in lowercase, such as `–delete` or `–lay`.
- Database names, layout names, and field names used in query strings are case insensitive, such as `–lay=mylayout` to specify `MyLayout`.

Note Field and database names that are used in XSLT statements outside of query strings are case sensitive and must exactly match the actual names used in the database. For example, in this statement:

```
<xsl:value-of select="fmrs:field[@name='LastName']"/>
```

the field reference “LastName” must exactly match the name of the LastName field in the database.

- Field names can contain periods, with the following exceptions:
 - 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 or XSLT 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,” below.

- 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`.

About the FileMaker Query Strings Reference

This release includes a FileMaker database called `Query Strings Reference.fp7` that contains brief descriptions and examples of each of the FileMaker query commands and query parameters. This can be found in the following directory on any machine in your FileMaker Server deployment (master or worker):

Mac:

```
/Library/FileMaker Server/Examples/XSLT
```

Windows:

```
<drive>:\Program Files\FileMaker\FileMaker Server\Examples\XSLT
```

Where: `<drive>` is the primary drive from which the system is started.

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.

For 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 `fmsresultset` grammar” on page 39.

For example, 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
```

Here is an example of 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. For example:

```
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.

For example, 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
```

For more information, see “`-delete.related` (Portal records delete) query parameter” on page 95.

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. For more information, see “`-relatedsets.filter` (Filter portal records) query parameter” on page 100 and “`-relatedsets.max` (Limit portal records) query parameter” on page 101.

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 89. 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 and requests made via XSLT stylesheets that don’t use sessions, the global values expire immediately after the request is made. For requests made via an XSLT stylesheet that use sessions, the global values persist for the duration of the session defined in the stylesheet, or until they are changed again with another request.

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.

For example:

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

Important If you are using a global field in an XSLT stylesheet, you must use the Admin Console to enable the XSLT Database Sessions option for the Web Publishing Engine. Otherwise, the values of global fields are not maintained between requests. See FileMaker Server Help.

Query command reference

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

Note For XSLT requests only, all of the following query commands require the `-grammar` query parameter.

-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 or XSLT.

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 as specified by `-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 –recid

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

Note For information on editing records in a portal, see “Editing records in a portal” on page 90.

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`

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.

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`

–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.
- The –findquery command, at the end of the complete statement.

For more information on using the –query parameter, see “–query (Compound find request) query parameter” on page 99.

–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 or XSLT

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 89.

Example:

To add a new record:

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

–process (Process XSLT stylesheets)

Processes an XSLT stylesheet without requesting data from the database. This query command can only be used with XSLT stylesheets.

Required query parameter: –grammar. You must use the fmresultset or FMPXMLRESULT grammar.

Example:

`http://192.168.123.101/fmi/xsl/my_template/my_stylesheet.xml?–grammar=fmresultset&–process`

See “Processing XSLT requests that do not query FileMaker Server” on page 58.

–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 or XSLT

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 recordset.

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 and XSLT requests. For information on parameters that are only available for XSLT requests, see “Using query strings in FileMaker XSLT stylesheets” on page 54.

–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 and –process

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 example 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`

–encoding (Encoding XSLT request) query parameter

Specifies the text encoding for an XSLT request. This query command can only be used with Custom Web Publishing with XSLT requests.

Value is: US-ASCII, ISO-8859-1, ISO-8859-15, ISO-2022-JP, Shift_JIS, or UTF-8

Optional with: all query commands in an XSLT request

Example:

`http://192.168.123.101/fmi/xsl/my_template/my_stylesheet.xml?–db=employees&–lay=departments
&–grammar=fmresultset&–encoding=Shift_JIS&–findall`

See “Setting text encoding for requests” on page 57.

–field (Container field name) query parameter

Specifies the name of a container field

Required with: request for data in a container field

See “About the URL syntax for FileMaker container objects in XML solutions” on page 36, and “About the URL syntax for FileMaker container objects in XSLT solutions” on page 53.

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. Field names can contain periods, with the following exceptions:

- 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 or XSLT 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 89.

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. The default operator is “begins with”. 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
```

You can use any FileMaker Pro find operator by specifying the `bw` keyword. For example, to find a range of values using the range operator (...), you would specify the `bw` keyword and then you would place the characters “...” before the search criteria.

Example:

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

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

–grammar (Grammar for XSLT stylesheets) query parameter

Specifies the grammar to use for an XSLT stylesheet. This query command can only be used with Custom Web Publishing with XSLT requests.

Value is: fmresultset, FMPXMLRESULT or FMPXMLLAYOUT

Required with: All XSLT requests

Example:

```
http://192.168.123.101/fmi/xsl/my_template/my_stylesheet.xml?–grammar=fmresultset&–db=mydatabase
&–lay=mylayout&–findall
```

See “Specifying an XML grammar for a FileMaker XSLT stylesheet” on page 54.

–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, –scriptnames, and –process (XSLT requests only).

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 or in XSLT stylesheet requests.

Value is: Name of the layout

Optional with: All query commands except –dbnames, –layoutnames, –scriptnames, and –process (XSLT requests only)

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 (which must be specified in lowercase). 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. The value `all` must be specified in lowercase. If `–max` is not specified, all records are returned.

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

Examples:

```
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 93.

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`.
- 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.

For example: `(q1);!(q2)`

In this example, `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>
```

For example:

```
–q1=typeofanimal&–q1.value=Cat
```

```
–q2=name&–q2.value=Fluffy
```

Example:

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

Example 1:

```
http://192.168.123.101/fmi/xml/fmresultset.xml?–db=employees&–lay=departments&–recid=22&–delete
```

Example 2:

```
http://localhost/fmi/xml/FMPXMLLAYOUT.xml?–db=test&–lay=empty&–view&–recid=9
```

–relatedsets.filter (Filter portal records) query parameter

Specifies whether to filter the rows returned for queries that use portal fields.

Value is : layout or none

- The default value is none if this parameter is not specified.
- If the query specifies layout, then the settings specified in the FileMaker Pro Portal Setup dialog are respected. The records are sorted based on the sort defined in the Portal Setup dialog, with the record set filtered to start with the initial row specified.
- If the Show Vertical Scroll Bar setting is enabled in the Portal Setup dialog, then you can use the –relatedsets.max option to specify the maximum number of rows to return in response to the query.
- If Show Vertical Scroll Bar setting is disabled, then the FileMaker Pro Portal Setup dialog’s Number of rows setting determines the number of rows to be displayed.
- If –relatedsets.filter is set to none, then the Web Publishing Engine returns all rows in the portal, and portal records that are not presorted.

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 rows to return in the results for this query.

Value is: an integer, or all.

- If the query specifies an integer, then the Web Publishing Engine returns that number of rows after the initial row are returned.
- If the query specifies all, then the Web Publishing Engine returns all of the related records.
- If the query does not specify the `-relatedsets.max` parameter, then the number of rows is determined by the value specified on the `-relatedsets.filter` parameter. See “`-relatedsets.filter (Filter portal records) query parameter`” on page 100.

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 48.

Value is: Script name

Optional with: all query commands except `-dbnames`, `-layoutnames`, `-process`, 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`, `–process`, 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`, `–process`, 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

In the following example, the first 10 records in the found set are skipped and records 11 through 15 are returned.

Example:

```
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 be 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 (starting with 1) that specifies the precedence to use for multiple sort fields.

In the following 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.

Example:

```
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 be 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

In the following 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.

Example:

```
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.

–stylehref (Style href) query parameter

Generates an XML-stylesheet processing instruction within the output document—setting the value of the href attribute (href=/mystylesheet.css or href=/stylesheets/mystylesheet.xml)—so you can use client-side, cascading stylesheets (CSS) or XSLT stylesheets with your XML document. The value of the –stylehref parameter must use an absolute path. The name of the stylesheet can be any name but it must contain an extension of either .css or .xml. See “Using server-side and client-side processing of stylesheets” on page 49. This parameter is used in conjunction with the –styletype parameter.

Optional with: All query commands

Requires: –styletype parameter

Example (assumes mystylesheet.xml is in the root folder of the web server software):

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

–styletype (Style type) query parameter

Generates an XML-stylesheet processing instruction within the output document—setting the value of the type attribute (type=text/css or type=text/xml)—so you can use client-side, cascading stylesheets (CSS) or XSLT stylesheets with your XML document. See “Using server-side and client-side processing of stylesheets” on page 49. This parameter is used in conjunction with the –stylehref parameter.

Optional with: All query commands

Requires: –stylehref parameter

Example (assumes mystylesheet.css is in the root folder of the web server software):

```
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=departments&-styletype=text/css
&-stylehref=/mystylesheet.css&-findall
```

–token.[string] (Pass values between XSLT stylesheets) query parameter

Passes any user-defined information between XSLT stylesheets without using sessions or cookies. This query parameter can only be used with Custom Web Publishing with XSLT requests.

string in –token.[string] is: Any alphanumeric string of any length, except blank spaces, including the numbers 0-9, lowercase letters a-z, or uppercase letters A-Z

User-defined parameter value is: Any character string that is URL encoded.

Optional with: All XSLT requests

Example:

```
http://192.168.123.101/fmi/xsl/template/my_stylesheet.xml?-db=employees&-lay=departments
&-grammar=fmresultset&-token.D100=Active&-findall
```

See “Using tokens to pass information between stylesheets” on page 59.

Appendix B

Error codes for Custom Web Publishing

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

- **Database and query string errors.** The Web Publishing Engine generates an error code for a published database whenever an XML data request occurs. See the next section, “Error code numbers for FileMaker databases.”
- **Web Publishing Engine errors.** When the Web Publishing Engine is in Development mode, it generates a specific error page when an error occurs in the Web Publishing Engine itself. When in Production mode, a general text message is displayed. See “Error code numbers for the Web Publishing Engine” on page 114.
- **FileMaker XSLT extension function errors.** You can use the `fmxslt:check_error_status()` extension function within an XSLT stylesheet to check the error status of extension functions when they are called. See “Error code numbers for the FileMaker XSLT extension functions” on page 115.

For a list of updated error codes, see the FileMaker Knowledge Base (<http://www.filemaker.com/kb/>).

Error code numbers for FileMaker databases

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 `FMPDSORESLT` grammars. An error code of 0 indicates that no error has occurred.

Here is an example of the database error code in the `fmresultset` grammar:

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

Here is an example of the database error code in the `FMPXMLRESULT` grammar:

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<!DOCTYPE FMPXMLRESULT PUBLIC "-//FMI//DTD FMPXMLRESULT//EN" "/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 Number	Description
-1	Unknown error
0	No error
1	User canceled action
2	Memory error
3	Command is unavailable (for example, wrong operating system, wrong mode, etc.)

Error Number	Description
4	Command is unknown
5	Command is invalid (for example, a Set Field script step does not have a calculation specified)
6	File is read-only
7	Running out of memory
8	Empty result
9	Insufficient privileges
10	Requested data is missing
11	Name is not valid
12	Name already exists
13	File or object is in use
14	Out of range
15	Can't divide by zero
16	Operation failed, request retry (for example, a user query)
17	Attempt to convert foreign character set to UTF-16 failed
18	Client must provide account information to proceed
19	String contains characters other than A-Z, a-z, 0-9 (ASCII)
20	Command or operation cancelled by triggered script
100	File is missing
101	Record is missing
102	Field is missing
103	Relationship is missing
104	Script is missing
105	Layout is missing
106	Table is missing
107	Index is missing
108	Value list is missing
109	Privilege set is missing
110	Related tables are missing
111	Field repetition is invalid
112	Window is missing
113	Function is missing
114	File reference is missing
115	The menu set is missing
116	The layout object is missing
117	The data source is missing
130	Files are damaged or missing and must be reinstalled
131	Language pack files are missing (such as template files)

Error Number	Description
200	Record access is denied
201	Field cannot be modified
202	Field access is denied
203	No records in file to print, or password doesn't allow print access
204	No access to field(s) in sort order
205	User does not have access privileges to create new records; import will overwrite existing data
206	User does not have password change privileges, or file is not modifiable
207	User does not have sufficient privileges to change database schema, or file is not modifiable
208	Password does not contain enough characters
209	New password must be different from existing one
210	User account is inactive
211	Password has expired
212	Invalid user account and/or password. Please try again
213	User account and/or password does not exist
214	Too many login attempts
215	Administrator privileges cannot be duplicated
216	Guest account cannot be duplicated
217	User does not have sufficient privileges to modify administrator account
300	File is locked or in use
301	Record is in use by another user
302	Table is in use by another user
303	Database schema is in use by another user
304	Layout is in use by another user
306	Record modification ID does not match
400	Find criteria are empty
401	No records match the request
402	Selected field is not a match field for a lookup
403	Exceeding maximum record limit for trial version of FileMaker Pro
404	Sort order is invalid
405	Number of records specified exceeds number of records that can be omitted
406	Replace/Reserialize criteria are invalid
407	One or both match fields are missing (invalid relationship)
408	Specified field has inappropriate data type for this operation
409	Import order is invalid
410	Export order is invalid
412	Wrong version of FileMaker Pro used to recover file
413	Specified field has inappropriate field type

Error Number	Description
414	Layout cannot display the result
415	One or more required related records are not available
416	A primary key is required from the data source table
417	The database is not a supported data source
500	Date value does not meet validation entry options
501	Time value does not meet validation entry options
502	Number value does not meet validation entry options
503	Value in field is not within the range specified in validation entry options
504	Value in field is not unique as required in validation entry options
505	Value in field is not an existing value in the database file as required in validation entry options
506	Value in field is not listed on the value list specified in validation entry option
507	Value in field failed calculation test of validation entry option
508	Invalid value entered in Find mode
509	Field requires a valid value
510	Related value is empty or unavailable
511	Value in field exceeds maximum number of allowed characters
512	The record was already modified by another user
513	To create a record, the record has to have a value in at least one field
600	Print error has occurred
601	Combined header and footer exceed one page
602	Body doesn't fit on a page for current column setup
603	Print connection lost
700	File is of the wrong file type for import
706	EPSF file has no preview image
707	Graphic translator cannot be found
708	Can't import the file or need color monitor support to import file
709	QuickTime movie import failed
710	Unable to update QuickTime file reference because the database file is read-only
711	Import translator cannot be found
714	Password privileges do not allow the operation
715	Specified Excel worksheet or named range is missing
716	A SQL query using DELETE, INSERT, or UPDATE is not allowed for ODBC import
717	There is not enough XML/XSL information to proceed with the import or export
718	Error in parsing XML file (from Xerces)
719	Error in transforming XML using XSL (from Xalan)
720	Error when exporting; intended format does not support repeating fields
721	Unknown error occurred in the parser or the transformer

Error Number	Description
722	Cannot import data into a file that has no fields
723	You do not have permission to add records to or modify records in the target table
724	You do not have permission to add records to the target table
725	You do not have permission to modify records in the target table
726	There are more records in the import file than in the target table. Not all records were imported
727	There are more records in the target table than in the import file. Not all records were updated
729	Errors occurred during import. Records could not be imported
730	Unsupported Excel version. Convert file to Excel 7.0 (Excel 95), 97, 2000, XP, or 2007 format and try again.
731	The file you are importing from contains no data
732	This file cannot be inserted because it contains other files
733	A table cannot be imported into itself
734	This file type cannot be displayed as a picture
735	This file type cannot be displayed as a picture. It will be inserted and displayed as a file
736	There is too much data to be exported to this format. It will be truncated
737	The Bento table you are importing is missing.
800	Unable to create file on disk
801	Unable to create temporary file on System disk
802	Unable to open file This error can be caused by one or more of the following: <ul style="list-style-type: none"> ■ Invalid database name ■ File is closed in FileMaker Server ■ Invalid permission
803	File is single user or host cannot be found
804	File cannot be opened as read-only in its current state
805	File is damaged; use Recover command
806	File cannot be opened with this version of FileMaker Pro
807	File is not a FileMaker Pro file or is severely damaged
808	Cannot open file because access privileges are damaged
809	Disk/volume is full
810	Disk/volume is locked
811	Temporary file cannot be opened as FileMaker Pro file
813	Record Synchronization error on network
814	File(s) cannot be opened because maximum number is open
815	Couldn't open lookup file
816	Unable to convert file
817	Unable to open file because it does not belong to this solution
819	Cannot save a local copy of a remote file
820	File is in the process of being closed

Error Number	Description
821	Host forced a disconnect
822	FMI files not found; reinstall missing files
823	Cannot set file to single-user, guests are connected
824	File is damaged or not a FileMaker file
825	File is not authorized to reference the protected file
900	General spelling engine error
901	Main spelling dictionary not installed
902	Could not launch the Help system
903	Command cannot be used in a shared file
904	Command can only be used in a file hosted under FileMaker Server
905	No active field selected; command can only be used if there is an active field
906	The current file is not shared; command can be used only if the file is shared
920	Can't initialize the spelling engine
921	User dictionary cannot be loaded for editing
922	User dictionary cannot be found
923	User dictionary is read-only
951	An unexpected error occurred
954	Unsupported XML grammar
955	No database name
956	Maximum number of database sessions exceeded
957	Conflicting commands
958	Parameter missing in query
959	Custom Web Publishing technology is disabled
1200	Generic calculation error
1201	Too few parameters in the function
1202	Too many parameters in the function
1203	Unexpected end of calculation
1204	Number, text constant, field name or "(" expected
1205	Comment is not terminated with "*/"
1206	Text constant must end with a quotation mark
1207	Unbalanced parenthesis
1208	Operator missing, function not found or "(" not expected
1209	Name (such as field name or layout name) is missing
1210	Plug-in function has already been registered
1211	List usage is not allowed in this function
1212	An operator (for example, +, -, *) is expected here
1213	This variable has already been defined in the Let function

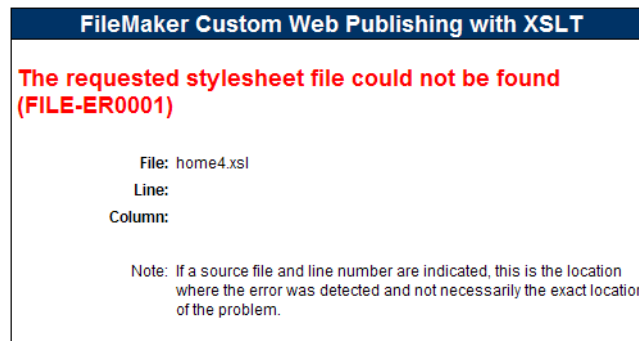
Error Number	Description
1214	AVERAGE, COUNT, EXTEND, GETREPETITION, MAX, MIN, NPV, STDEV, SUM and GETSUMMARY: expression found where a field alone is needed
1215	This parameter is an invalid Get function parameter
1216	Only Summary fields allowed as first argument in GETSUMMARY
1217	Break field is invalid
1218	Cannot evaluate the number
1219	A field cannot be used in its own formula
1220	Field type must be normal or calculated
1221	Data type must be number, date, time, or timestamp
1222	Calculation cannot be stored
1223	The function referred to is not yet implemented
1224	The function referred to does not exist
1225	The function referred to is not supported in this context
1400	ODBC client driver initialization failed; make sure the ODBC client drivers are properly installed.
1401	Failed to allocate environment (ODBC)
1402	Failed to free environment (ODBC)
1403	Failed to disconnect (ODBC)
1404	Failed to allocate connection (ODBC)
1405	Failed to free connection (ODBC)
1406	Failed check for SQL API (ODBC)
1407	Failed to allocate statement (ODBC)
1408	Extended error (ODBC)
1409	Extended error (ODBC)
1410	Extended error (ODBC)
1411	Extended error (ODBC)
1412	Extended error (ODBC)
1413	Extended error (ODBC)
1450	Action requires PHP privilege extension
1451	Action requires that current file be remote
1501	SMTP authentication failed
1502	Connection refused by SMTP server
1503	Error with SSL
1504	SMTP server requires the connection to be encrypted
1505	Specified authentication is not supported by SMTP server
1506	Email message(s) could not be sent successfully
1507	Unable to log in to the SMTP server

Error code numbers for the Web Publishing Engine

When the Web Publishing Engine is in Development mode, it generates a specific error page when an error occurs in the Web Publishing Engine itself. This type of error may result from a variety of causes, such as if the Web Publishing Engine is unable to:

- find a requested or nested (via <xsl:include>) stylesheet file
- parse a requested or nested stylesheet file because of an XML error with the file
- generate a stylesheet from the file because of an XSLT or XPath error in the file
- process the request because the XML grammar is not properly specified in the CGI
- communicate with the Web Publishing Core to retrieve XML

When the Web Publishing Engine is operating in Development mode, the error page for this type of error contains an error message and error number in parentheses. For example:



Example of error page when the Web Publishing Engine is in Development mode

Here is a partial list of the Web Publishing Engine error code values:

Error code value	Description
QUERY-ER0001	No XML grammar was specified in the -grammar query parameter
QUERY-ER0002	“xxx” is not a valid XML grammar for FileMaker XSLT
FILE-ER0001	The requested stylesheet file could not be found
FILE-ER0002	The requested file could not be found
UNKNOWN	An unexpected error has occurred
MCS-000 to MCS-600	An unexpected error has occurred
MCS-601	The resource “x” could not be loaded because there is no support for resources of type: “x”
MCS-602	The URL “x” could not be resolved
MCS-603	The HTTP request for “x” returned an error of type “x”
MCS-604	The resource “x” could not be loaded because of an unexpected error
MCS-605	The resource “x” could not be loaded because the content-type was invalid
MCS-606	The resource “x” could not be loaded because of an XML error in the document
MCS-607	The resource “x” could not be loaded because of an authentication problem
MCS-700	An unexpected error has occurred
MCS-800	An unexpected error has occurred

If the Web Publishing Engine is in Production mode, the following default general text message in the `pe_server_error.html` error page is displayed for Web Publishing Engine errors:

An unexpected error has occurred while using FileMaker Custom Web Publishing with XSLT.

The default `pe_server_error.html` file contains the preceding text message in six languages.

You, as the developer, can edit the text of the `pe_server_error.html` error page as necessary for your solution. The `pe_server_error.html` file is located in the `cwpe` folder inside the `publishing-engine` folder on the host where you installed the Web Publishing Engine.

For information on how to set the Web Publishing Engine into Development or Production mode, see FileMaker Server Help.

Error code numbers for the FileMaker XSLT extension functions

The `fmxml:check_error_status()` extension function (see “Checking the error status of extension functions” on page 76), returns one of the errors in the following table:

Error code value	Description
-1	Unknown error
0	No error
General Errors	
10000	Invalid header name
10001	Invalid HTTP status code
Session Errors	
10100	Unknown session error
10101	Requested session name is already used
10102	Session could not be accessed - maybe it does not exist
10103	Session has timed out
10104	Specified session object does not exist
Messaging Errors	
10200	Unknown messaging error
10201	Message formatting error
10202	Message SMTP fields error
10203	Message “To Field” error
10204	Message “From Field” error
10205	Message “CC Field” error
10206	Message “BCC Field” error
10207	Message “Subject Field” error
10208	Message “Reply-To Field” error

Error code value	Description
10209	Message body error
10210	Recursive mail error - attempted to call send_email() inside an email XSLT stylesheet
10211	SMTP authentication error - either login failed or wrong type of authentication provided
10212	Invalid function usage - attempted to call set_header(), set_status_code() or set_cookie() inside an email XSLT stylesheet
10213	SMTP server is invalid or is not working.
Formatting Errors	
10300	Unknown formatting error
10301	Invalid date time format
10302	Invalid date format
10303	Invalid time format
10304	Invalid day format
10305	Improperly formatted date time string
10306	Improperly formatted date string
10307	Improperly formatted time string
10308	Improperly formatted day string
10309	Unsupported text encoding
10310	Invalid URL encoding
10311	Regular expression pattern error

Index

A

- access log files for web server, described 84
- access privileges 20
- accounts and privileges
 - enabling for Custom Web Publishing 19
 - Guest account 20
 - scripts 22
- Admin Console for Web Publishing Engine 35
- Administration Console for Web Publishing Engine 27
- application log 76, 84
- ASCII characters, in XML documents 46
- authenticated base URI parameter 61
- authentication of web users 19
 - basic 61
 - password 62
- auto-enter attribute 40
- available scripts 94

B

- base URI parameter 61
- Basic Authentication for web users 19, 61
- break_encode() extension function 69
- buffering, using in stylesheet 63

C

- Change Password script 20
- check_error_status() extension function 76, 115
- checkboxes, checking for values in 70
- client information, obtaining via XSLT parameters 60
- client-side stylesheets 35, 49
- commands for queries. *See* query strings
- compare_date() extension function 72
- compare_datetime() extension function 73
- compare_day() extension function 73
- compare_time() extension function 73
- comparing strings 70
- comparison operators for fields 97
- compound find query command 93
- compound find query parameter 99

- container fields
 - how web users access data 21
 - publishing contents of 21, 30
 - URL syntax for accessing in XML solutions 36
 - URL syntax for accessing in XSLT solutions 53
- contains_checkbox_value() extension function 70
- content buffering, using 63
- convert_datetime() extension function 73
- cookies
 - extension functions, using 68
 - storing session ID 64
- create_session() extension function 64
- creating a new record 93
- Custom Web Publishing
 - access to solutions by web users 19, 61
 - definition 11
 - described 15, 16
 - enabling in database 19
 - enabling in Web Publishing Engine 20
 - extended privilege for 19
 - Guest account 20
 - new features in 16
 - overview 11
 - requirements for 17
 - restricting IP address access in web server 20
 - scripts 23
 - using a static IP address 17
 - using scripts 22
 - using XML 33
 - using XSLT 26, 51
 - with PHP 13
 - with XML 13
 - with XSLT 13
- Custom Web Publishing Engine (CWPE) 26, 34

D

- database error codes 38
- database layouts available 93
- database sessions, enabling 65, 91
- databases, protecting when published 20
- <datasource> element 39
- date extension functions, using 72
- date format strings 73
- day extension functions, using 72
- db query parameter 95
- dbnames query command 91

- defining extension functions 76
- delete query command 91
- delete.related query parameter 90
- deleting portal records 90
- Development mode, Web Publishing Engine 114
- document type definitions (DTDs) 38, 42
- document() function 61
- documentation 9
- documentation information 9, 18
- documents, loading via document() function 61
- dup query command 92

E

- edit query command 92
- electronic documentation 9
- elements
 - database error code 38
 - in FMPXMLLAYOUT grammar 43
 - in FMPXMLRESULT grammar 42
 - in fmresultset grammar 39
- email messages
 - extension functions for 66
 - initial default encoding setting 57
- enabling Custom Web Publishing in database 19
- encoding
 - encoding query parameter 57, 95
 - output via <xsl:output> element 58
 - requests 57
 - URLs 37, 64
 - using string manipulation extension functions 69
 - XML data 38, 46
 - XSLT stylesheets 58
- encoding query parameter 95
- <error code> and <ERRORCODE> elements 107
- errors
 - about error codes 107
 - checking error status of extension functions 76, 115
 - database error code elements 38
 - database error code numbers 107
 - extension function error code numbers 115
 - log files for web server 84
 - pe_application_log.txt log file 84
 - pe_server_error.html error page 115
 - Web Publishing Engine error code numbers 114
- examples of
 - generated FMPXMLLAYOUT grammar 45
 - generated FMPXMLRESULT grammar 43
 - generated fmresultset grammar 41

- export XML data 33
- extended privilege for Custom Web Publishing 19
- Extensible Markup Language (XML). *See* XML
- extension functions for FileMaker XSLT
 - See also* fmxslt extension functions

F

- field name query parameter (non-container) 96
- field names, fully qualified syntax 89
- field query parameter (container) 95
- <field-definition> element 40
- fieldname.op query parameter 97
- FileMaker API for PHP 13
 - definition 13
- FileMaker Pro, contrast with Web Publishing Engine 33
- FileMaker Server
 - documentation 9
 - installing 9
- FileMaker Server Admin
 - see 20
- FileMaker Server Admin Console 35
- FileMaker Site Assistant. *See* XSLT Site Assistant
- FileMaker-specific XSLT parameters 59
- filtering data with stylesheets 25
- filtering portal field rows 100
- find query command 92
- findall query command 92
- findany query command 92
- findquery query command 93
- FMPDSORESLT grammar
 - compared to other grammars 37
- FMPXMLLAYOUT grammar 33, 43–45
 - compared to other grammars 37
- FMPXMLRESULT grammar 33, 42–43
 - compared to other grammars 37
- fmresultset grammar 33, 39–41
 - compared to other grammars 37
- fmxslt keyword for enabling XML publishing 19, 35

fmxslt extension functions

- fmxslt:break_encode() function 69
- fmxslt:check_error_status() function 76, 115
- fmxslt:compare_date() function 72
- fmxslt:compare_datetime() function 73
- fmxslt:compare_day() function 73
- fmxslt:compare_time() function 73
- fmxslt:contains_checkbox_value() function 70
- fmxslt:convert_datetime() function 73
- fmxslt:create_session() function 64
- fmxslt:get_cookie() function 68
- fmxslt:get_cookies() function 68
- fmxslt:get_date() function 72
- fmxslt:get_datetime() function 73
- fmxslt:get_day() function 72
- fmxslt:get_fm_date_format() function 72
- fmxslt:get_fm_time_format() function 72
- fmxslt:get_fm_timestamp_format() function 72
- fmxslt:get_header() function 67
- fmxslt:get_long_date_format() function 72
- fmxslt:get_long_day_format() function 72
- fmxslt:get_long_time_format() function 72
- fmxslt:get_session_object() function 65
- fmxslt:get_short_date_format() function 72
- fmxslt:get_short_day_format() function 72
- fmxslt:get_short_time_format() function 72
- fmxslt:get_time() function 72
- fmxslt:html_encode() function 69
- fmxslt:invalidate_session() function 64, 65
- fmxslt:regex_contains() function 70
- fmxslt:remove_session_object() function 65
- fmxslt:send_email() functions 66
- fmxslt:session_encode_url() function 64
- fmxslt:session_exists() function 64
- fmxslt:set_cookie() function 68
- fmxslt:set_header() function 67
- fmxslt:set_session_object() function 65
- fmxslt:set_session_timeout() function 64
- fmxslt:set_status_code() function 67
- fmxslt:url_decode() function 70
- fmxslt:url_encode() function 69

fmxslt keyword for enabling XSLT publishing 19, 27

format strings, date and time 73

formatting data with stylesheets 25

four-digit-year attribute 40

fully qualified field name, syntax of 89

G

generating a static page 58

get_cookie() extension function 68

get_cookies() extension function 68

get_date() extension function 72

get_datetime() extension function 73

get_day() extension function 72

get_fm_date_format() extension function 72

get_fm_time_format() extension function 72

get_fm_timestamp_format() extension function 72

get_header() extension function 67

get_long_date_format() extension function 72

get_long_day_format() extension function 72

get_long_time_format() extension function 72

get_session_object() extension function 65

get_short_date_format() extension function 72

get_short_day_format() extension function 72

get_short_time_format() extension function 72

get_time() extension function 72

GIF files, publishing on web 21

global attribute 40

global fields

- database sessions, enabling 65, 91

- syntax of 91

- using with sessions 65, 91

–grammar query parameter 54, 98

grammars for XML, described 37

grammars recommended for XSLT 54

Guest account

- disabling 20

- enabling 20

- with Custom Web Publishing 20

H

header functions, using 67

hiding metadata with stylesheets 25

HTML

- forms for XML requests 35

- reformatting XML data into 33

html_encode() extension function 69

I

import XML data 33

installation documentation 9

Instant Web Publishing

- definition 11

- documentation 9

integrating data with stylesheets 25

invalidate_session() extension function 64, 65

ISO-2022-JP encoding 57

ISO-8859-1 encoding 57
 ISO-8859-15 encoding 57

J

JavaScript
 defining extension functions 76
 JDBC documentation 9
 JPEG files, publishing on web 21
 jsessionid parameter 64

K

keywords for enabling Custom Web Publishing 19, 27, 35

L

–lay query parameter 48, 98
 –lay.response query parameter 48, 98
 layout information, using in stylesheet 62
 –layoutnames query command 93
 layouts, switching for an XML response 48
 limiting portal field rows 101
 loading additional documents 61
 log files 82, 85
 described 83
 logging via <xsl:message> element 76
 pe_application_log.txt 84
 pe_internal_access_log.txt 85
 web server access 84
 web_server_module_log.txt 84
 <xsl:message> element 84
 Logs folder 84
 –lop query parameter 98

M

mail messages. *See* email messages
 –max query parameter 99
 max-characters attribute 40
 max-repeat attribute 40
 metadata, hiding with stylesheets 25
 <metadata> element 40
 method attribute, <xsl:output> element 58
 MIME (Multipurpose Internet Mail Extensions)
 types 20
 –modid query parameter 99
 monitoring websites 83

N

name attribute 40
 namespaces for
 XML 38
 XSLT 55
 new features in Custom Web Publishing 16
 –new query command 93
 not-empty attribute 40
 numbers for
 database error codes 107
 extension function error codes 115
 Web Publishing Engine error codes 114
 numeric-only attribute 40

O

ODBC documentation 9
 online documentation 9
 operators, comparison 97
 order of XML request processing 48
 output pages
 encoding, specifying 58
 initial default encoding setting 57
 output method, specifying 58
 <xsl:output> element 58
 outputting data with stylesheets 25
 overview
 Custom Web Publishing 11
 overview of steps for
 XML data access 35
 XSLT publishing 27

P

parameters for queries. *See* query strings
 parameters for XSLT, FileMaker-specific 59
 passing information between stylesheets 59
 passwords
 access to XML documents 62
 Basic Authentication for web users 19, 61
 Change Password script 20
 defining for Custom Web Publishing 19
 no login password 20
 PDFs 9
 pe_application_log.txt log file 84
 pe_internal_access_log.txt log file 85
 pe_server_error.html error page 115
 Perl regular expressions, comparing strings 70

PHP

- advantages 14
- troubleshooting 31

PHP API for Custom Web Publishing 13

portal field queries 100, 101

portals

- adding records 89
- deleting records 90
- editing records 90
- layout 100
- sorting records 100

privilege set, assigning for Custom Web Publishing 19

–process query command 58, 94

processing a Web Publishing Engine request 12

processing XSLT stylesheets 94

Production mode, Web Publishing Engine 115

protecting published databases 20

publishing on the web

- connecting to Internet or intranet 17
- container field objects 21, 30
- database error codes 107
- protecting databases 20
- QuickTime movies 21
- requirements for 17
- using XML 15, 35
- using XSLT 16, 27, 51

Q

query information, accessing in request 60

–query query parameter 99

query strings 46, 54, 87

- adding records to portals 89
- commands and parameters 46, 54, 87
- editing records in portals 90
- fully qualified field name, syntax of 89
- global fields, syntax of 91
- guidelines for 88
- Query Strings Reference 88
- requesting XML data 46, 87
- statically defined in XSLT stylesheets 55
- XSLT stylesheets, using in 54

Query Strings Reference 88

querying portal fields 90

QuickTime movies, publishing on the web 21

R

–recid query parameter 100

regex_contains() extension function 70

<relatedset-definition> element 40

–relatedsets.filter query parameter 100

–relatedsets.max query parameter 101

Re-Login script 20

remove_session_object() extension function 65

requests for XML data 35

requirements for Custom Web Publishing 17

result attribute 40

<resultset> element 40

retrieving available script names 94

retrieving layout information 94

retrieving layout names 93

S

SAT

see Admin Console 20

Scalable Vector Graphics (SVG), transforming XML data into 33

–script query parameter 101

–script.param query parameter 101

–script.prefind query parameter 102

–script.prefind.param query parameter 102

–script.presort query parameter 102

–script.presort.param query parameter 103

–scriptnames query command 94

scripts

accounts and privileges 22

Change Password 20

database sessions, enabling 65

for XML requests 35

in Custom Web Publishing 22

Re-Login 20

tips and considerations 22

triggers 23

security

accounts and passwords 20

documentation 12

guidelines for protecting published databases 20

restricting access from IP addresses 20

statically defined query strings, using 55

send_email() extension functions 66

server-side XSLT stylesheets 25, 51

session extension functions, use in stylesheets 64

session_encode_url() extension function 64

session_exists() extension function 64

set_cookie() extension function 68

set_header() extension function 67

set_session_object() extension function 65

set_session_timeout () extension function 64
 set_status_code() extension function 67
 Shift_JIS encoding 57
 Site Assistant
 described 16
 Site Assistant. *See* XSLT Site Assistant
 –skip query parameter 103
 –sortfield query parameter 104
 –sortorder query parameter 104
 specifying layout when requesting XML data 48
 specifying XML grammar 54
 SSL (Secure Sockets Layer) encryption 20
 state, saving in sessions 64
 static publishing, definition 11
 statically defined query strings in XSLT
 stylesheets 55
 storing information in sessions 64
 strings
 comparing via Perl regular expressions 70
 using string manipulation extension functions 69
 –stylehref query parameter 105
 stylesheets
 about 25
 checkboxes, checking for values in 70
 client-side 49
 comparing strings via Perl regular expressions 70
 content buffering, using 63
 cookie extension functions 68
 date and time format strings 73
 date, time, and day extension functions 72
 developing 51
 email messages, sending 66
 encoding of 58
 error status of extension functions, checking 76
 examples of usage 25
 –grammar parameter for 54
 guidelines for developing 51
 header functions, using 67
 query strings for 54
 server-side 25, 51
 session function, using 64
 string manipulation extension functions 69
 testing 82, 83
 using in website or program 30
 using layout information in 62
 using XSLT Site Assistant to create 28
 XML-stylesheet processing instruction 49
 XSLT, described 25
 –styletype query parameter 105

summary of steps for
 XML data access 35
 XSLT publishing 27
 switching layout for XML response 48
 switching layouts for an XML response 48

T

technology tests 31
 testing
 websites 82
 XML output 83
 testing PHP publishing 31
 text encoding
 –encoding query parameter 57, 95
 encoding settings 57
 for XSLT requests 57
 generated XML data 38
 initial default settings 57
 request and output pages default 57
 URLs 37, 64
 using string manipulation extension functions 69
 time extension functions, using 72
 time format strings 73
 time-of-day attribute 40
 –token query parameter 59, 105
 tool for XSLT, described 16
 tools for XSLT, described 28
 transforming data with stylesheets 25
 triggers 23
 troubleshooting
 Custom Web Publishing websites 82
 XML document access 50
 XSLT stylesheets 31
 type attribute 40

U

Unicode characters 46
 URL syntax for
 container objects in XML solutions 36
 container objects in XSLT solutions 53
 XML requests 35
 XSLT stylesheets 52
 URL text encoding 37
 url_decode() extension function 70
 url_encode() extension function 69
 US-ASCII encoding 57

- user names
 - access to XML documents 62
 - Basic Authentication for web users 19, 61
 - defining for Custom Web Publishing 19
- User-Agent header, checking 59
- UTF-8 (Unicode Transformation 8 Bit)
 - encoding setting 57
 - format 37, 46

V

- values, checking for in checkboxes 70
- vCards, reformatting XML data into 33
- view query command 94

W

- web browsers
 - receiving output 12
 - role in XML requests 34
 - role in XSLT-CWP requests 26
- Web folder, copying container field objects 21
- Web Publishing Core
 - illustrated 26, 34
 - internal access logs 85
- Web Publishing Engine
 - Admin Console 35
 - Administration Console 27
 - application log 84
 - benefits of 15
 - described 12
 - Development mode 114
 - generated error codes 107
 - generating pages from XSLT stylesheet 26
 - generating XML data 34
 - generating XML documents 35
 - Production mode 115
 - request processing 12
- web server
 - log files 84
 - MIME type support 20
 - role in XML requests 34
 - role in XSLT-CWP requests 26
- web users
 - accessing protected databases 19, 61, 62
 - requirements for accessing Custom Web Publishing solutions 17
 - using container field data 21
- web_server_module_log.txt log file 84

- websites
 - creating with Web Publishing Engine 15
 - FileMaker support pages 9
 - monitoring 83
 - testing 82
- wpc_access_log.txt file 85

X

XML

- described 33
- document type definitions (DTDs) 38, 39, 42
- enabling in database 19
- encoded using UTF-8 format 38, 46
- filtering data 33
- FMPXMLLAYOUT grammar 43
- FMPXMLRESULT grammar 42
- fmresultset grammar 39
 - <datasource> element 39
 - <field-definition> element 40
 - <metadata> element 40
 - <relatedset-definition> element 40
 - <resultset> element 40
- generating XML data from request 34
- grammars, described 37
- namespaces for 38
- order of request processing 48
- parsers 35, 46
- query strings 46, 87
- requesting data 35
- summary of steps for accessing XML data 35
- troubleshooting access to XML documents 50
- URL text encoding 37
- using client-side stylesheets 49
- XML 1.0 specification 33
- XML-stylesheet processing instruction 49

XML and XSLT advantages 14

XML custom web publishing 13

XML request

- specifying layout 48

XML response

- switching layout 48

XPath statements 59

- <xsl:stylesheet> element 55, 59, 60, 83
- <xsl:message> element 76
- <xsl:output> element 58
- <xsl:param> element 59
 - <xsl:param name="authenticated-xml-base-uri"/> parameter 61
 - <xsl:param name="client-ip"/> parameter 60
 - <xsl:param name="client-password"/>

parameter 60

`<xsl:param name="client-user-name"/>`

parameter 60

`<xsl:param name="request-query"/>` parameter 60

`<xsl:param name="xml-base-uri"/>` parameter 61

`<xsl:template>` element 60, 61, 83

`<xsl:variable>` element 61

XSLT

checkboxes, checking for values in 70

comparing strings via Perl regular expressions 70

content buffering, using 63

cookie extension functions 68

date and time format strings 73

date, time, and day extension functions 72

described 25

developing stylesheets 51

email messages, sending 66

enabling in database 19

error status of extension functions, checking 76

examples of stylesheets 25

extension functions for FileMaker 59

FileMaker-specific XSLT parameters 59

generating pages from XSLT stylesheet 26

–grammar parameter 54

header functions, using 67

JavaScript extensions 76

layout information, using 62

namespaces for 55

query strings for 54

Query Strings Reference 88

server-side stylesheets 25, 51

string manipulation extension functions 69

summary of steps for publishing 27

troubleshooting stylesheets 31

using stylesheets in website or program 30

XSLT 1.0 specification 25

XSLT Site Assistant, using 28

XSLT-CWP requests 26

xslt-template-files folder 27, 30, 62

XSLT custom web publishing 13

XSLT Site Assistant

described 28

generated stylesheets, described 29

preparing for use 28

starting 29

using 29

`<?xslt-cwp-buffer buffer-content="true"?>` processing instruction 63

`<?xslt-cwp-query?>` processing instruction 51, 55

xslt-template-files folder 27, 30, 62