## Contents

**Chapter 1**  
*Introducing Custom Web Publishing*

- About this guide  
- Creating dynamic web sites with the Web Publishing Engine  
  - About Custom Web Publishing with XML  
  - About Custom Web Publishing with XSLT  
  - About the tools for developing XSLT stylesheets  
- Web publishing requirements  
  - What is required to publish a database using Custom Web Publishing  
  - What web users need to access a Custom Web Publishing solution  
  - Connecting to the Internet or an intranet  
- New features in Custom Web Publishing with XML and XSLT  
- FileMaker scripts and Custom Web Publishing  
  - Script tips and considerations  
- Migrating web publishing solutions from previous versions of FileMaker Pro  
- Where to go from here

**Chapter 2**  
*Preparing databases for Custom Web Publishing*

- Enabling Custom Web Publishing in a database  
- When web users use Custom Web Publishing to access a protected database  
- Protecting your published databases  
- Web server support for Internet media types (MIME)  
- About publishing the contents of container fields on the web  
  - Publishing container field objects stored in a database  
  - Publishing container field objects stored as a file reference  
  - How web users can use container field data

**Chapter 3**  
*Accessing XML data with the Web Publishing Engine*

- Using Custom Web Publishing with XML  
  - Differences between the Web Publishing Engine and FileMaker Pro XML Import/Export  
  - How the Web Publishing Engine generates XML data from a request  
- General process for accessing XML data from the Web Publishing Engine  
- About the URL syntax for XML data and container objects  
  - What’s new in the URL syntax for XML data  
  - About the URL syntax for XML data  
  - About the URL syntax for FileMaker container objects in XML solutions  
  - About URL text encoding
Accessing XML data via the Web Publishing Engine
  - About namespaces for FileMaker XML
  - About FileMaker database error codes
  - Retrieving the document type definitions for the FileMaker grammars
Using the fmresultset grammar
  - Description of elements in the fmresultset grammar
  - Example of XML data in the fmresultset grammar
Using other FileMaker XML grammars
  - Description of elements in the FMPXMLRESULT grammar
  - Example of XML data in the FMPXMLRESULT grammar
  - Description of elements in the FMPXMLLAYOUT grammar
  - Example of XML data in the FMPXMLLAYOUT grammar
About UTF-8 encoded data
Using FileMaker query strings to request XML data
Switching layouts for an XML response
Understanding how an XML request is processed
Using server-side and client-side processing of stylesheets
Troubleshooting XML document access

Chapter 4
Introduction to Custom Web Publishing with XSLT

About FileMaker XSLT stylesheets
  - What are some examples of using FileMaker XSLT stylesheets?
  - Getting started using Custom Web Publishing with XSLT
  - How the Web Publishing Engine generates pages based on XML data and XSLT stylesheets
General steps for using Custom Web Publishing with XSLT
Using the FileMaker Site Assistant to generate FileMaker XSLT stylesheets
  - Installing the Site Assistant
  - Before using the Site Assistant
  - Starting the Site Assistant
  - Using the Site Assistant
  - About the Site Assistant’s generated stylesheets
Using the FileMaker CDML Converter
  - About the FileMaker CDML Converter
  - Installing the CDML Converter
  - Starting and using the CDML Converter
  - Checking and fixing the CDML Converter’s generated stylesheets
  - Using the CDML Converter’s generated stylesheets
  - Testing the CDML Converter’s generated stylesheets
Chapter 5

Developing FileMaker XSLT stylesheets

Using XSLT stylesheets with the Web Publishing Engine
About the FileMaker XSLT Extension Function Reference
About the URL syntax for FileMaker XSLT stylesheets
About the URL syntax for FileMaker container objects in XSLT solutions
Using query strings in FileMaker XSLT stylesheets
Specifying an XML grammar for a FileMaker XSLT stylesheet
About namespaces and prefixes for FileMaker XSLT stylesheets
Using statically defined query commands and query parameters
Setting text encoding for requests
Specifying an output method and encoding
About the encoding of XSLT stylesheets
Processing XSLT requests that do not query FileMaker Server
Using tokens to pass information between stylesheets
Using the FileMaker XSLT extension functions and parameters
   About the FileMaker-specific XSLT parameters set by the Web Publishing Engine
   Accessing the query information in a request
   Obtaining client information
   Using the Web Publishing Engine base URI parameter
   Using the authenticated base URI parameter
   Loading additional documents
   Using a database’s layout information in a stylesheet
   Using content buffering
   Using Web Publishing Engine sessions to store information between requests
   Using the session extension functions
   Sending email messages from the Web Publishing Engine
   Using the header functions
   Using the cookie extension functions
   Using the string manipulation extension functions
   Comparing strings using Perl 5 regular expressions
   Checking for values in a field formatted as a checkbox
   Using the date, time, and day extension functions
   Checking the error status of extension functions
   Using logging
Chapter 6

Testing and monitoring a site

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

Appendix A

Valid names used in query strings

About the query commands and parameters
  - Obsolete request names and parameters
  - Guidelines for using query commands and parameters
  - About the FileMaker Query Strings Reference
  - About the syntax for a fully qualified field name
  - Adding records to a portal
  - Editing records in a portal
  - About the syntax for specifying a global field

Using the query commands
  - `–dbnames` (Database names) query command
  - `–delete` (Delete record) query command
  - `–dup` (Duplicate record) query command
  - `–edit` (Edit record) query command
  - `–find`, `–findall`, or `–findany` (Find records) query commands
  - `–layoutnames` (Layout names) query command
  - `–new` (New record) query command
  - `–process` (Process XSLT stylesheets)
  - `–scriptnames` (Script names) query command
  - `–view` (View layout information) query command

Using the query parameters
  - `–db` (Database name) query parameter
  - `–encoding` (Encoding XSLT request) query parameter
  - `–field` (Container field name) query parameter
  - `fieldname` (Non-container field name) query parameter
  - `fieldname.op` (Comparison operator) query parameter
  - `–grammar` (Grammar for XSLT stylesheets) query parameter
  - `–lay` (Layout) query parameter
  - `–lay.response` (Switch layout for response) query parameter
  - `–lop` (Logical operator) query parameter
Chapter 1
Introducing Custom Web Publishing

With FileMaker® Server 7 Advanced, you can publish FileMaker databases on the Internet or an intranet in these ways:

- Custom Web Publishing using Extensible Markup Language (XML)
- Custom Web Publishing using Extensible Stylesheet Language Transformations (XSLT) stylesheets

Custom Web Publishing with XML and XSLT gives you choices and control over the design and functionality of your web pages. FileMaker Server, which hosts the published databases, does not require FileMaker Pro to be installed or running for Custom Web Publishing to be available.

To support Instant Web Publishing and Custom Web Publishing with XML and XSLT, 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. The Web Publishing Engine functions as an XSLT processor and provides output as HTML, XML, or text (such as vCards) to the web server, which then provides the output to the web browser.

Web users access your Custom Web Publishing solution either by clicking an HREF link or entering a Uniform Resource Locator (URL) that specifies the web server address and a FileMaker query string request. The URL can either 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 stylesheets.

Important  Security is increasingly important when you publish data on the web. Review the security guidelines in the FileMaker Security Guide, located in the Electronic Documentation folder (inside the English Extras folder).
**About this guide**

This guide assumes you are experienced with XML and XSLT, developing web sites, and using FileMaker Pro to create databases. 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
- how to convert CDML solutions to FileMaker XSLT

**Important** You can download PDFs of FileMaker 7 documentation from www.filemaker.com. Any updates to this document are also available from the web site.

The documentation for FileMaker Server also includes the following additional information:

### For information about

|---|---|
| Installing and configuring FileMaker Server | FileMaker Server Administrator’s Guide  
FileMaker Server Administration Help |
| Instant Web Publishing | FileMaker Instant Web Publishing Guide |
| Publishing files from previous versions of FileMaker Pro | Converting FileMaker Databases from Previous Versions |

**Creating dynamic web sites 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 web sites and applications.

- **Data integration:** By using FileMaker XSLT stylesheets, you can integrate any subset of the FileMaker data into other web sites and with other middleware and custom applications. You can make the data look like it belongs to another web site instead of displaying an entire FileMaker layout in the web browser.

- **Security:** As the administrator for the Web Publishing Engine, you 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 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.

Assisted migration from CDML solutions: The FileMaker CDML Converter tool helps you convert CDML format files to XSLT stylesheets and is useful in your process of learning XSLT. CDML format files and XSLT stylesheets are similar and the differences are easily understandable. XSLT stylesheets can be more complex and powerful than CDML format files.

About Custom Web Publishing with XML

Custom Web Publishing with XML provides the ability to query for data in FileMaker databases, and to easily use the data in whatever way you want to. 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 3, “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 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.

About the tools for developing XSLT stylesheets

FileMaker Server Advanced includes two tools for developing XSLT stylesheets:

- The FileMaker Site Assistant is an application you can use to create basic XSLT stylesheets as a starting point for Custom Web Publishing with XSLT. The 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 the FileMaker Site Assistant to generate FileMaker XSLT stylesheets” on page 40.

- The FileMaker CDML Converter is an application that converts existing CDML format files to XSLT stylesheets that are compatible with Custom Web Publishing with XSLT. It is a useful tool for beginning the process of CDML website migration and learning how FileMaker XSLT stylesheets are constructed. See “Using the FileMaker CDML Converter” on page 42.
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, the following are required:
- a Windows- or Mac OS X-based computer running FileMaker Server, enabled for Custom Web Publishing
- one or more FileMaker Pro databases hosted by FileMaker Server
- the FileMaker Server Web Publishing Engine, installed and configured
- a web server, either Microsoft IIS on Windows or Apache on Mac OS X
- the IP address or domain name of the host running the web server
- any web browser and access to the web server to develop and test your Custom Web Publishing solution

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:
- any web browser software
- 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:
- It is strongly recommended that you 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 users when your computer is connected to the Internet or an intranet.
- The host computer for the Web Publishing Engine 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 users to locate your databases. If you are not sure of the type of access available to you, consult your ISP or network administrator.

New features in Custom Web Publishing with XML and XSLT

Custom Web Publishing with XML and XSLT provides several important new features:
- Databases are hosted on FileMaker Server, and FileMaker Pro is not required to be running.
- You can now use server-side XSLT stylesheet processing, which is more secure than client-side stylesheet processing.
- 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 51.
Like FileMaker Pro, access to data, layouts, and fields is based on the users’s 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 18.

Web users can perform complex, multi-step scripts. About 70 script steps are now supported on the web. See the next section, “FileMaker scripts and Custom Web Publishing.”

The new 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.

Some new query commands and parameters have been added, and some have become obsolete. To access data in a database, you must specify a layout. For security reasons, accessing data without specifying a layout is no longer supported. 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 78.

**FileMaker scripts and Custom Web Publishing**

The ScriptMaker™ feature in FileMaker Pro is useful for automating frequently performed tasks, or for combining several tasks. When used with Custom Web Publishing, FileMaker scripts can allow web users to perform more tasks or a series of tasks.

FileMaker supports about 70 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 the Indicate web compatibility checkbox in the Edit Script dialog box 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. Before sharing your database, evaluate all scripts that will be executed from a web browser. Be sure to log in with different user accounts to make sure they work as expected for all clients.

Keep these tips and considerations in mind:

- Use accounts and privileges to restrict the set of scripts that a web user can execute. Verify that the scripts contain only web-compatible script steps, and only provide access to scripts that should be used from a web browser.

- Consider the side effects of scripts that execute a combination of steps that are controlled by access privileges. For example, if a script includes a step to delete records, and a web user does not log in with an account that allows record deletion, the script will not execute the Delete Records script step. However, the script might continue to run, which could lead to unexpected results.

- In scripts, you can 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 will stop the script from continuing.
- If Allow User Abort is off, unsupported script steps are skipped over 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 will 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.

Similarly, any script that modifies data should include the Commit Record/Request step, because all 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, it’s a good idea to 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 7.0v1” you know that the current user is accessing your database with Custom Web Publishing. For more information on functions, see FileMaker Pro Help.

After converting your files, you should open each script that web users might run and select Indicate web compatibility to ensure that the script will execute properly with Custom Web Publishing.

If you are using a script in an XSLT stylesheet that sets or modifies a state, you must use the Administration Console to enable the XSLT Database Sessions option for the Web Publishing Engine. Otherwise, states are not maintained between requests. See the FileMaker Server Advanced Web Publishing Installation Guide.

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

<table>
<thead>
<tr>
<th>Script step</th>
<th>Behavior in Custom Web Publishing solutions</th>
</tr>
</thead>
<tbody>
<tr>
<td>Perform Script</td>
<td>Scripts cannot perform in other files, unless the files are hosted on FileMaker Server and Custom Web Publishing is enabled in the other files.</td>
</tr>
<tr>
<td>Exit Application</td>
<td>Logs off web users, closes all windows, but does not exit the web browser application.</td>
</tr>
<tr>
<td>Allow User Abort</td>
<td>Determines how unsupported script steps are handled. Enable to stop scripts from continuing, and disable to skip over unsupported steps. See above for more details.</td>
</tr>
<tr>
<td>Set Error Capture</td>
<td>This is always enabled with Custom Web Publishing, and prevents web users from aborting scripts.</td>
</tr>
<tr>
<td>Pause/Resume script</td>
<td>Although this script is supported in Custom Web Publishing, it should be avoided. When a pause step is executed the script runtime pauses. Only a script containing the resume script step can make it resume execution. If the script runtime remains in a paused state until the session times out, then the script will not be completed.</td>
</tr>
<tr>
<td>Sort Records</td>
<td>A specified sort order must be saved with the script step to execute in Custom Web Publishing.</td>
</tr>
<tr>
<td>Open URL</td>
<td>This script has no effect in a Custom Web Publishing solution.</td>
</tr>
</tbody>
</table>
Introducing Custom Web Publishing

Migrating web publishing solutions from previous versions of FileMaker Pro

Before updating existing databases to FileMaker Pro 7, it is important to review the Converting FileMaker Databases from Previous Versions guide, located in the Electronic Documentation folder (inside the English Extras folder).

When migrating the web publishing solution, start by backing up your files and setting up a separate test environment to do the conversion. After developing the XML or XSLT solution, test its functionality and security (accounts and privileges) before placing the site into production.

Here are some other considerations when migrating solutions you want to publish using XML or XSLT:

- The access privileges model has been improved. Consider re-assigning user names and passwords to take advantage of accounts and privileges. See FileMaker Pro Help.
- Plug-ins designed for use with FileMaker Pro are not automatically enabled for FileMaker Server. See the FileMaker Server Advanced Web Publishing Installation Guide.
- The Web Security Databases are no longer supported. If you relied on them for web-based security, you must transfer the accounts, passwords, and associated privileges into your converted database files in FileMaker Pro. See the Converting FileMaker Databases from Previous Versions Guide.
- If you are migrating a CDML solution, see appendix C, “Converting CDML solutions to FileMaker XSLT.”
- If your database has a container field that stores a file reference instead of an actual object, 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 19.
- The URL syntax has changed for web-published files. If you created any links to access a database on the web, you must update the links using the new syntax and database location after converting the file and hosting it on FileMaker Server. See “About the URL syntax for XML data and container objects” on page 23, and “About the URL syntax for FileMaker XSLT stylesheets” on page 48.
- Script support for the web has been enhanced.
  - Access privileges for web users can be set to allow execution of specific, individual scripts and prevent execution of others.
  - ScriptMaker script steps are now clearly indicated as “web compatible”; special scripts can be created for web users. After converting your files, open each script that web users might run and enable Indicate web compatibility to see if your scripts contain steps that might produce unexpected results when executed from a web browser.
  - On the web, scripts are always executed with Error Capture on. If your scripts relied on Error Capture being disabled (off), they may not function the same way. Test your files thoroughly after conversion.
Where to go from here

Here are some suggestions for what to do and where to find information in the documentation to get started developing Custom Web Publishing solutions:


- 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 17.

- To learn how to access data in FileMaker databases using XML, see chapter 3, “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 2**  
*Preparing databases for Custom Web Publishing*

Before you can use Custom Web Publishing with a database, there are some steps you must take to 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 either 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. Create a new extended privilege using one of the following keywords for the type of Custom Web Publishing technology you want to allow for the database:
   - To allow Custom Web Publishing with XML, enter this keyword: `fmxml`
   - To allow Custom Web Publishing with XSLT, enter this keyword: `fmxslt`

   For information on how to create a new extended privilege, see FileMaker Pro Help.
3. Assign the new extended privilege to one or more privilege sets.
4. Assign the privilege set(s) that include the Custom Web Publishing extended privilege to one or more accounts, or to the Admin or Guest account.

**Note** When defining account names and passwords for Custom Web Publishing solutions, use printable ASCII characters, for example a-z, A-Z, and 0-9. For more secure account names and passwords, include punctuation characters such as “!” and “%,” but do not include colons. For information on setting up accounts, see FileMaker Pro Help.

**When web users use Custom Web Publishing to access 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.

Here is a summary of what happens 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.

Web users generally cannot modify their account password from a web browser. It is possible, however, to build this functionality into your database with the Change Password script step (to enable web users to change their password). 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.
- To enable or disable a type of Custom Web Publishing technology for all Custom Web Publishing solutions in the Web Publishing Engine, use the Administration Console. See the FileMaker Server Advanced Web Publishing Installation Guide.
- You can 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.
- You can 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 the use of mathematical formulas known as ciphers. These ciphers are then 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 published databases, see the FileMaker Security Guide.

**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 either 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 24, and “About the URL syntax for FileMaker container objects in XSLT solutions” on page 48.

Publishing container field objects stored as a file reference

If a container field stores file references instead of actual files, then you must do the following steps to use the Web Publishing Engine 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: <root 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 can use 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.
- 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.
Chapter 3

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, then 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-specific 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 5, “Developing FileMaker XSLT stylesheets.”

For more general information on XML, additional examples that use XML, and links to XML resources, see the FileMaker web site 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 FMPDSORESULT grammar. See “Accessing XML data via the Web Publishing Engine” on page 25.
- 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 Web Publishing Engine Administration Console, make sure XML Publishing is enabled. See the *FileMaker Server Advanced Web Publishing Installation Guide*.

2. In FileMaker Pro, open each FileMaker database that you’re publishing and make sure the database has the fmxml extended privilege enabled for XML Custom Web Publishing. See “Enabling Custom Web Publishing in a database” on page 17.
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.

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 33, 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 35.

### 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 48 and “About the URL syntax for FileMaker container objects in XSLT solutions” on page 48.

### What’s new in the URL syntax for XML data

The Web Publishing Engine uses a new URL syntax for accessing XML data.

- FileMaker Pro 6 and earlier versions used the following syntax in requests for XML data:
  
  ```xml
  FMPro?<CGI_request>
  ```

  This syntax has changed. See the next section, “About the URL syntax for XML data.”

- The –lay query parameter for specifying a database layout is now required with all query commands except –dbnames, –layoutnames, and –scriptnames. See appendix A, “Valid names used in query strings.”

- The –format parameter is obsolete. The XML grammar for XML requests is now specified in the URL syntax before the query string. See the next section, “About the URL syntax for XML data.”

**Note** Unlike XML requests, the grammar for XSLT stylesheets is specified with the –grammar query parameter. See “Specifying an XML grammar for a FileMaker XSLT stylesheet” on page 50.

### About the URL syntax for XML data

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

```xml
<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, FMPXMLEVENT.xml, or FMPDSORESULT.xml. See “Using the fmresultset grammar” on page 27 and “Using other FileMaker XML grammars” on page 29.

<query string> is a combination of one query command and one or more query parameters for FileMaker XML. (The –dbnames command doesn’t require any parameters.) See “Using FileMaker query strings to request XML data” on page 33, 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, FMPXMLEVENT, and FMPDSORESULT. For information on the rules for case sensitivity of the query string, see “Guidelines for using query commands and parameters” on page 76.

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


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 77.

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 19.

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 48.

### 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 33.

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
```

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 fmresultset grammar” on page 27.

- **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 29.

- **FMPDSORESULT**: The FMPDSORESULT grammar, which is supported in FileMaker Pro for exporting XML, is deprecated for accessing XML data via the Web Publishing Engine. The FMPDSORESULT grammar is not supported for XSLT stylesheets. For information on the FMPDSORESULT 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 33.

### 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.

<table>
<thead>
<tr>
<th>For this grammar</th>
<th>This default namespace is generated</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmresultset</td>
<td>xmlns=&quot;<a href="http://www.filemaker.com/xml/fmresultset">http://www.filemaker.com/xml/fmresultset</a>&quot;</td>
</tr>
<tr>
<td>FMPXMLRESULT</td>
<td>xmlns=&quot;<a href="http://www.filemaker.com/fmpxmlresult">http://www.filemaker.com/fmpxmlresult</a>&quot;</td>
</tr>
<tr>
<td>FMPXMLLAYOUT</td>
<td>xmlns=&quot;<a href="http://www.filemaker.com/fmpxmllayout">http://www.filemaker.com/fmpxmllayout</a>&quot;</td>
</tr>
</tbody>
</table>

### 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.

<table>
<thead>
<tr>
<th>For this grammar</th>
<th>This syntax is used</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmresultset</td>
<td>&lt;error code=&quot;0&quot;&gt;&lt;/error&gt;</td>
</tr>
<tr>
<td>FMPXMLRESULT</td>
<td>&lt;ERRORCODE&gt;0&lt;/ERRORCODE&gt;</td>
</tr>
<tr>
<td>FMPDSORESULT</td>
<td>&lt;ERRORCODE&gt;0&lt;/ERRORCODE&gt;</td>
</tr>
</tbody>
</table>

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.

<table>
<thead>
<tr>
<th>For this grammar</th>
<th>Use this HTTP request</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmresultset</td>
<td>http://&lt;host&gt;[:&lt;port&gt;/fmi/xml/fmresultset.dtd</td>
</tr>
<tr>
<td>FMPXMLRESULT</td>
<td>http://&lt;host&gt;[:&lt;port&gt;/fmi/xml/FMPXMLRESULT.dtd</td>
</tr>
<tr>
<td>FMPXMLLAYOUT</td>
<td>http://&lt;host&gt;[:&lt;port&gt;/fmi/xml/FMPXMLLAYOUT.dtd</td>
</tr>
<tr>
<td>FMPDSORESULT</td>
<td>http://&lt;host&gt;[:&lt;port&gt;/fmi/xml/FMPDSORESULT.dtd?db=&lt;database&gt;&amp;-lay=&lt;layout&gt;</td>
</tr>
</tbody>
</table>
**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**

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

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Full form</th>
<th>Short form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Year</td>
<td>yyyy (4 digits)</td>
<td>yy (2 digits)</td>
</tr>
<tr>
<td>Month</td>
<td>MM (2 digits)</td>
<td>M (1 or 2 digits)</td>
</tr>
<tr>
<td>Day</td>
<td>dd (2 digits)</td>
<td>d (1 or 2 digits)</td>
</tr>
</tbody>
</table>

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

<table>
<thead>
<tr>
<th>Field</th>
<th>Full form</th>
<th>Short form</th>
</tr>
</thead>
<tbody>
<tr>
<td>Hour (1 – 12)</td>
<td>hh (2 digits)</td>
<td>h (1 or 2 digits)</td>
</tr>
<tr>
<td>Hour (1 – 24)</td>
<td>kk (2 digits)</td>
<td>k (1 or 2 digits)</td>
</tr>
<tr>
<td>Minute</td>
<td>mm</td>
<td></td>
</tr>
<tr>
<td>Second</td>
<td>ss</td>
<td></td>
</tr>
<tr>
<td>AM/PM</td>
<td>a</td>
<td></td>
</tr>
</tbody>
</table>

The `<metadata>` element of the fmresultset grammar contains one or more `<field-definition>` and `<related-set-definition>` elements, each containing attributes for one of the fields of the result set. These attributes specify whether the field is an auto-enter field (“yes” or “no”), the maximum number of repeating values (max-repeat attribute), whether it is a not-empty field (“yes” or “no”), whether it is a global field (“yes” or “no”), result (“text,” “number,” “date,” “time,” “timestamp,” or “container”), 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.

The `<resultset>` element contains all of 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
<?xml version="1.0" 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>
  <product build="01/10/2004" name="FileMaker Web Publishing Engine" version="7.0v1" />
  <datasource database="art" date-format="M/d/yyyy" layout="web" table="art" time-format="h:mm:ss a" total-count="3" />
  <metadata>
    <field-definition auto-enter="no" global="no" max-repeat="1" name="Title" not-empty="no" result="text" type="normal" />
    <field-definition auto-enter="no" global="no" max-repeat="1" name="Artist" not-empty="no" result="text" type="normal" />
    <field-definition auto-enter="no" global="no" max-repeat="1" name="Image" not-empty="no" result="container" type="normal" />
    <field-definition auto-enter="no" global="no" max-repeat="2" name="Description" not-empty="no" result="text" type="normal" />
    <relatedset-definition table="artlocations">
      <field-definition auto-enter="no" global="no" max-repeat="1" name="artlocations::Location" not-empty="no" result="text" type="normal" />
      <field-definition auto-enter="no" global="no" max-repeat="1" name="artlocations::Date" not-empty="no" result="text" type="normal" />
    </relatedset-definition>
    <field-definition auto-enter="no" global="no" max-repeat="1" name="artistbio::birthdate" not-empty="no" result="date" type="normal" />
  </metadata>
  <resultset count="1" fetch-size="1">
    <record mod-id="5" record-id="1">
      <field name="Title">
        <data>Village Market</data>
      </field>
      <field name="Artist">
        <data>Camille Pissarro</data>
      </field>
    </record>
  </resultset>
</fmresultset>
```
<field name="Image">
    <data>fmi/xml/cnt/data.jpg?-db=art&amp;-lay=web&amp;-recid=1&amp;-field=Image(1)</data>
</field>
<field name="Description">
    <data>Impressionist</data>
    <data>French</data>
</field>
<relatedset count="2" table="artlocations">
    <record mod-id="0" record-id="3">
        <field name="artlocations::Location">
            <data>Home</data>
        </field>
        <field name="artlocations::Date">
            <data>1/1/2001</data>
        </field>
    </record>
    <record mod-id="0" record-id="4">
        <field name="artlocations::Location">
            <data>Restoration</data>
        </field>
        <field name="artlocations::Date">
            <data>12/15/2001</data>
        </field>
    </record>
</relatedset>
<field name="artistbio::birthdate">
    <data>7/10/1830</data>
</field>
</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 27.

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 85), 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
<?xml version="1.0" standalone="no"?>
<!DOCTYPE FMPXMLRESULT PUBLIC "-//FMI//DTD FMPXMLRESULT//EN" "fmi/xml/FMPXMLRESULT.dtd">
<FMPXMLRESULT xmlns="http://www.filemaker.com/fmpxmlresult">
  <ERRORCODE>0</ERRORCODE>
  <PRODUCT BUILD="01/10/2004" NAME="FileMaker Web Publishing Engine" VERSION="7.0v1" />
  <DATABASE DATEFORMAT="M/d/yyyy" LAYOUT="web" NAME="art" RECORDS="3" TIMEFORMAT="h:mm:ss a" />
  <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" />
    <FIELD EMPTYOK="YES" MAXREPEAT="2" NAME="Description" TYPE="TEXT" />
    <FIELD EMPTYOK="YES" MAXREPEAT="1" NAME="artlocations::Location" TYPE="TEXT" />
  </METADATA>
</FMPXMLRESULT>
```
<FIELD EMPTYOK="YES" MAXREPEAT="1" NAME="artlocations::Date" TYPE="TEXT" />
<FIELD EMPTYOK="YES" MAXREPEAT="1" NAME="artistbio::birthdate" TYPE="DATE" />
</METADATA>
<RESULTSET FOUND="1">
  <ROW MODID="5" RECORDID="1">
    <COL>
      <DATA>Village Market</DATA>
    </COL>
    <COL>
      <DATA>Camille Pissarro</DATA>
    </COL>
    <COL>
      <DATA>/fmi/xml/cnt/data.jpg?-db=art&amp;-lay=web&amp;-recid=1&amp;-field=Image(1)
      </DATA>
    </COL>
    <COL>
      <DATA>Impressionist</DATA>
      <DATA>French</DATA>
    </COL>
    <COL>
      <DATA>Home</DATA>
      <DATA>Restoration</DATA>
    </COL>
    <COL>
      <DATA>1/1/2001</DATA>
      <DATA>12/15/2001</DATA>
    </COL>
    <COL>
      <DATA>7/10/1830</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.
**Example of XML data in the FMPXMLLAYOUT grammar**

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

```xml
<?xml version="1.0" standalone="no"?>
<!DOCTYPE FMPXMLLAYOUT PUBLIC "-//FMI//DTD FMPXMLLAYOUT//EN" "fmi/xml/FMPXMLLAYOUT.dtd">
<FMPXMLLAYOUT xmlns="http://www.filemaker.com/fmpxmllayout">
  <ERRORCODE>0</ERRORCODE>
  <PRODUCT BUILD="01/10/2004" NAME="FileMaker Web Publishing Engine" VERSION="7.0v1"/>
  <LAYOUT DATABASE="art" NAME="web">
    <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="Description">
      <STYLE TYPE="POPUPMENU" VALUELIST="keywords"/>
    </FIELD>
    <FIELD NAME="artlocations::Location">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
    <FIELD NAME="artlocations::Date">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
    <FIELD NAME="artistbio::birthdate">
      <STYLE TYPE="EDITTEXT" VALUELIST=""/>
    </FIELD>
  </LAYOUT>
  <VALUELISTS>
    <VALUELIST NAME="keywords">
      <VALUE>Impressionist</VALUE>
      <VALUE>Cubist</VALUE>
      <VALUE>Abstract</VALUE>
      <VALUE>French</VALUE>
      <VALUE>Italian</VALUE>
      <VALUE>Spanish</VALUE>
      <VALUE>American</VALUE>
    </VALUELIST>
  </VALUELISTS>
</FMPXMLLAYOUT>
```
About UTF-8 encoded data

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

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

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

The UTF-8 encoding format includes the following features:

- All ASCII characters are one-byte UTF-8 characters. A legal ASCII string is a legal UTF-8 string.
- Any non-ASCII character (any character with the high-order bit set) is part of a multibyte character.
- The first byte of any UTF-8 character indicates the number of additional bytes in the character.
- The first byte of a multibyte character is easily distinguished from the subsequent byte, which makes it 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”:

```
```

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 –dbname 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 5, “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 49.

<table>
<thead>
<tr>
<th>Use this query command name</th>
<th>To execute this command</th>
</tr>
</thead>
<tbody>
<tr>
<td>–dbnames</td>
<td>Retrieve names of all hosted and web-shared databases</td>
</tr>
<tr>
<td>–delete</td>
<td>Delete record</td>
</tr>
<tr>
<td>–dup</td>
<td>Duplicate record</td>
</tr>
<tr>
<td>–edit</td>
<td>Edit record</td>
</tr>
<tr>
<td>–find</td>
<td>Find record(s)</td>
</tr>
<tr>
<td>–findall</td>
<td>Find all records</td>
</tr>
</tbody>
</table>
### Use this query command name | To execute this command
--- | ---
–findany | Find a random record
–layoutnames | Retrieve names of all available layouts for a hosted and web-shared database
–new | New record
–scriptnames | Retrieve names of all available scripts for a hosted and web-shared database
–view | Retrieves layout information from a database if the FMPXMLLAYOUT grammar is specified. Retrieves <metadata> section of XML document and an empty recordset if the fmresultset or FMPXMLRESULT grammar is specified.

### Use these query parameter names | With these query commands
--- | ---
–db (database name) | Required with all query commands except –dbnames and –process (XSLT requests only)
–field | Required to specify a field in a URL for container requests. See “About the URL syntax for FileMaker container objects in XML solutions” on page 24.
fieldname | At least one field name is required with –edit. Optional with –find. See “fieldname (Non-container field name) query parameter” on page 82.
fieldname.op (operator) | Optional with –find
–lay (layout name) | Required with all query commands, except –dbnames, –layoutnames, –scriptnames, and –process (XSLT requests only)
–lay.response (switch layout for XML response) | Optional with all query commands, except –dbnames, –layoutnames, –scriptnames, and –process (XSLT requests only)
–lop (logical operator) | Optional with –find
–max (maximum records) | Optional with –find, –findall
–modid (modification ID) | Optional with –edit
–script.presort (perform script before sort) | Optional with –find, –findall
–skip (skip records) | Optional with –find, –findall
–sortfield.[1-9] (sort field) | Optional with –find, –findall
–sortorder.[1-9] (sort order) | Optional with –find, –findall
–stylehref (stylesheet HREF) | Optional with all query commands (to specify a stylesheet URL for –styletype)
–styletype (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 –script.prefind query parameter, if specified.
2. Process the query commands, such –find or –new.
4. Sort the resulting data, if a sort was specified.
5. Process the –lay.response query parameter to switch to a different layout, if this is specified.
7. 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 5,
“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/xsl).
- 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.xsl. The name of the stylesheet can be any name but it must contain an extension of either .css
or .xsl.

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/xsl
&-stylehref=/mystylesheet.xsl

**Note** This “/” in “-stylehref=/document.xsl” 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
<?xml-stylesheet type="text/xsl" href="/mystylesheet.xsl"?>
```

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 web site or
program” on page 45.

**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 17.
- The database is hosted and opened by FileMaker Server. See FileMaker Server Administration Help.
- The database account name and password you are using, if any, are correct.
- The web server and the Web Publishing Engine are running.
- XML Publishing is enabled in the Web Publishing Engine. See the FileMaker Server Advanced Web
Publishing Installation Guide.
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 two tools to help you get started creating XSLT stylesheets—the Site Assistant and the CDML Converter. For more detailed information about how FileMaker XSLT stylesheets are constructed, see chapter 5, “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 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 web sites 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 54.

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, such as how to use the FileMaker-specific XSLT extension functions and query commands and parameters. The Site Assistant and the CDML Converter are tools to help you get started creating 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 Web Publishing Engine Administration Console, make sure XSLT Publishing is enabled. See the FileMaker Server Advanced Web Publishing Installation Guide.

2. Using FileMaker Pro, open each FileMaker database that you’re publishing and make sure the database has the fmxslt extended privilege enabled for Custom Web Publishing with XSLT. See “Enabling Custom Web Publishing in a database” on page 17.

   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 the FileMaker Site Assistant tool to create one or more basic XSLT stylesheets as a starting point for your site. See the next section, “Using the FileMaker Site Assistant to generate FileMaker XSLT stylesheets.”

   If you have existing CDML solutions, you can use the CDML Converter to convert the CDML format files to XSLT stylesheets. See “Using the FileMaker CDML Converter” on page 42.

   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 5, “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 7 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 web site or program” on page 45.

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

   For example, you can use a static page such as index.html for the web site that either 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. See the FileMaker Security Guide.

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 the FileMaker Site Assistant to generate FileMaker XSLT stylesheets

The FileMaker 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. The 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 the Site Assistant to edit or update existing stylesheets, but you can use the 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 the 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 the 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 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 XML export.

Installing the Site Assistant

For information about installing the Site Assistant, see the *FileMaker Server Advanced Web Publishing Installation Guide*.

Before using the Site Assistant

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

- Set the extended privilege “fmxslt” in the database. Use privilege sets when running the Site Assistant that are equivalent to those you give to web users. See “Enabling Custom Web Publishing in a database” on page 17.
- Open and host the database in FileMaker Server. See FileMaker Server Administration Help.
- Be sure the web server and the Web Publishing Engine are running.
Starting the Site Assistant

To start the Site Assistant, do one of the following:

- Double-click the FileMaker Site Assistant application icon.
- Windows: Click the Start button, and choose FileMaker Site Assistant from the Programs menu.

Using the Site Assistant

For detailed information and step-by-step procedures for using the Site Assistant, see the Site Assistant Help. For information about using the Site Assistant’s generated stylesheets, see “Using FileMaker XSLT stylesheets in a web site or program” on page 45.

Important When using the 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 the Site Assistant’s generated stylesheets

The XSLT stylesheets generated by the 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 the Site Assistant. See “Using statically defined query commands and query parameters” on page 51.
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 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 55.

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 56.

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

For information about other sections of the Site Assistant stylesheets, see chapter 5, “Developing FileMaker XSLT stylesheets.”

### Using the FileMaker CDML Converter

In this release, XSLT replaces CDML (a FileMaker proprietary markup language) as the Custom Web Publishing language for FileMaker databases. FileMaker no longer supports CDML. To migrate web sites from CDML format files to FileMaker XSLT stylesheets, use the FileMaker CDML Converter.

**About the FileMaker CDML Converter**

The FileMaker CDML Converter is an application that converts working CDML format files to XSLT stylesheets that are compatible with Custom Web Publishing with XSLT. It is a useful tool for beginning the process of CDML web site migration and learning how FileMaker XSLT stylesheets are constructed.

Before using the CDML Converter, it is recommended that you copy the CDML format files from the production environment to a temporary working directory. After using the CDML Converter, you will need to examine the generated stylesheets and conversion log. In some cases, the converted XSLT stylesheets are ready for use with FileMaker Server without any further modification. In other cases, the CDML Converter may be unable to convert all of the CDML tags to XSLT stylesheets and you will need to do some manual editing of the converted XSLT stylesheets. You can use your own XSLT stylesheet authoring or text editing tools to modify the stylesheets as necessary. Some changes to the database may also be required if CDML format files are referenced in database records. This is because the names of the converted XSLT stylesheets will use the .xsl filename extension.

You can only use the CDML Converter to convert working CDML format files into new stylesheets. You cannot use it to edit converted XSLT stylesheets. To use the CDML Converter and migrate the CDML web site to XSLT effectively, you should be an experienced CDML developer and you should be familiar with Custom Web Publishing with XSLT.

**Note** The generated XSLT stylesheets transform FileMaker data into HTML pages based on the fmresultset XML grammar.

### Installing the CDML Converter

For information about installing the CDML Converter, see the *FileMaker Server Advanced Web Publishing Installation Guide.*
**Starting and using the CDML Converter**

**Important** It is recommended that you first copy all of the files from the cdml_format_files folder (if you chose to use it) and all of the files from your FileMaker Pro Web folder to a temporary working folder, and do the conversion processing on the copied files.

**Note** When converting a source file to an XSLT stylesheet, the CDML Converter keeps the same filename but changes the filename extension to .xsl. If the source folder contains multiple files with the same name and different extensions, such as Myfile.html and Myfile.cdml, a file overwriting problem will occur during conversion. For example, the CDML Converter converts Myfile.html to Myfile.xsl, and then attempts to convert Myfile.cdml to Myfile.xsl. Because Myfile.xsl already exists, the CDML Converter displays a dialog box to confirm that you want to overwrite it. To avoid this problem before conversion, make sure the files in the source folder have unique filenames. If you change any filenames, be sure you change any references to those files before conversion.

To start and use the CDML Converter:

1. To start the CDML Converter, do one of the following:
   - Double-click the FileMaker CDML Converter application icon.
   - Windows: Click the **Start** button, and choose **FileMaker CDML Converter** from the **Programs** menu.

2. For **Source Folder**, click **Select** to specify the folder where the CDML source files are stored.
3. Locate and select the folder that contains the CDML source files, then click **Select**.
4. For **Text encoding of files**, choose the encoding of the CDML source format files. See “Setting text encoding for requests” on page 52.
   **Note** All source format files in a conversion session must have the same text encoding.
5. For **Destination Folder**, click **Select** to specify the folder where you want to store the converted XSLT stylesheets and other files from the CDML web site.
   The CDML Converter creates a folder hierarchy and a set of files in the destination folder that correspond to the source files. The CDML Converter copies files from the source folder that do not require CDML conversion, such as the GIF files in an images subfolder, without changing them. The converted files have the same filename, but the filename extension is changed to .xsl.
6. Locate and select the folder where you want to save the XSLT stylesheets on your hard disk, then click Select.

7. Click Start.

   The CDML Converter converts the CDML format files and saves a conversion log with the converted XSLT stylesheets in the destination folder you specified. The conversion log is also shown in the CDML Converter window.

8. Click Done.

Checking and fixing the CDML Converter’s generated stylesheets

After using the CDML Converter, you can obtain error information about the conversion process by reviewing the conversion log that is shown in the CDML Converter window, or by opening the conversion log file in the destination folder. The conversion log file is named cdml2xsl_<datetime>.log, where <datetime> is the date and time at the start of the conversion.

If a warning or error occurs during the conversion, the CDML Converter adds a message to the conversion log and inserts an XSLT comment tag into the converted XSLT stylesheet with a description of the problem. The errors and warnings will be indicated by one of the following in XSLT comment tags in the converted XSLT stylesheet:

<!-- CDML Converter ERROR: <description of error> -->
<!-- CDML Converter WARNING: <description of warning> -->

The CDML Converter uses a specified set of CDML to XSLT mapping rules to convert the CDML format files. All conversion errors result from situations where the CDML Converter could not automatically determine the correct CDML to XSLT conversion. For all conversion errors, you must either manually fix the cause of the error in the CDML file and then convert the file again, or use a text editor or XSLT stylesheet editor to manually fix the problems in the converted XSLT stylesheets.

Here are some common types of conversion errors:

- A CDML tag is not supported in Custom Web Publishing with XSLT. For example, the –dbclose CDML action tag is not supported. To fix this type of error, you must either change the CDML tag to use a supported tag, remove the functionality from the CDML format file, or fix the logic in the XSLT stylesheet.

- A CDML tag is not recognized. For example, a CDML tag might be misspelled. To fix this type of error, you must either change the CDML tag to use a valid syntax, remove the tag, or fix the logic in the XSLT stylesheet.

- A CDML parameter is not recognized. For example, the CDML List parameter in the [FMP-ValueList: Field Name, List=Value List Name] CDML tag is not supported in XSLT stylesheets. To fix this type of error, you must either change the CDML parameter to use a valid syntax, remove the parameter, or fix the logic in the XSLT stylesheet.

For more information about the CDML to XSLT mapping rules and other situations where you will have to manually fix XSLT statements because the CDML tag conversion could not be done, see appendix C, “Converting CDML solutions to FileMaker XSLT.”

Notes

- If there are any query parameters that reference CDML format files from database fields, you will need to manually update the references in the database. If any such references are found in any of the converted XSLT stylesheets, a note is inserted in the XSLT stylesheet reminding you to make the changes in the database.
If the converted XSLT stylesheets include a FileMaker date or time extension function, such as fmxslt:get_date(), then the function uses the configured locale for FileMaker Server. See “Using the date, time, and day extension functions” on page 66. After conversion, you must manually change and localize date and time formatting strings that are passed to these functions, such as changing Month/Day/Year to Day/Month/Year.

**Using the CDML Converter’s generated stylesheets**

After fixing the errors in the converted XSLT stylesheets, you can use them with the Web Publishing Engine. See the following section, “Using FileMaker XSLT stylesheets in a web site or program.”

**Testing the CDML Converter’s generated stylesheets**

You should thoroughly test the converted XSLT stylesheets before using them in a production environment. See chapter 6, “Testing and monitoring a site.”

**Using FileMaker XSLT stylesheets in a web site or program**

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

To use FileMaker XSLT stylesheets in a web site 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 7 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 `<img src="logo.jpg">`. 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 19.

   **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 48.
Note For web sites, it’s 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. The Site Assistant can create a home.xsl 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.xsl 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.xsl

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 17.
- The database is hosted and opened by FileMaker Server. See FileMaker Server Administration Help.
- The database account name and password you are using, if any, are correct.
- The web server and the Web Publishing Engine are running.
Chapter 5

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 48.
- The stylesheet filename and the folder name where the stylesheet is stored must be URL-encoded UTF-8. 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 50.
- 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 48 and “Using statically defined query commands and query parameters” on page 51.
- 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 52.
- 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 its default text encoding setting for output pages. See “Specifying an output method and encoding” on page 53.
- 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 61.

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

The FileMaker Server Advanced CD includes a FileMaker database called XSLT Reference.fp7 that contains brief descriptions and examples of each of the FileMaker XSLT extension functions. You can open the FileMaker XSLT Extension Function Reference in the Custom Web Publishing Reference folder on the FileMaker Server Advanced CD.

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.xsl>[?<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.xsl>` 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 49, 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 51.

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 76.

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.xsl?-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 77.

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 19.

- 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 `<xsl-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 33.

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

<table>
<thead>
<tr>
<th>Use this XSLT query command or parameter name</th>
<th>To</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>--grammar</td>
<td>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.”</td>
<td>This query parameter is required in all XSLT requests.</td>
</tr>
<tr>
<td>--encoding</td>
<td>Specify the text encoding for a request. See “Setting text encoding for requests” on page 52.</td>
<td>This query parameter is optional in all XSLT requests.</td>
</tr>
<tr>
<td>--process</td>
<td>Process a stylesheet without requesting data. See “Processing XSLT requests that do not query FileMaker Server” on page 53.</td>
<td>This query command requires the --grammar query parameter.</td>
</tr>
<tr>
<td>--token</td>
<td>Pass values between pages without using sessions or cookies. See “Using tokens to pass information between stylesheets” on page 54.</td>
<td>This query parameter is optional in all XSLT requests.</td>
</tr>
</tbody>
</table>
**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 fmresultset grammar” on page 27. You can also use the older FMPXMLRESULT or FMPXMLLAYOUT grammars. To access value lists and field display information in layouts, you must use the FMPXMLLAYOUT grammar. See “Using other FileMaker XML grammars” on page 29. You cannot use the FMPDSORESULT 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:

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

Or

<?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.

<table>
<thead>
<tr>
<th>If you use this</th>
<th>Declare this namespace</th>
<th>Use this prefix</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmresultset XML grammar</td>
<td>xmlns:fmrs=&quot;<a href="http://www.filemaker.com/xml/fmresultset">http://www.filemaker.com/xml/fmresultset</a>&quot;</td>
<td>fmrs</td>
</tr>
<tr>
<td>FMPXMLRESULT grammar</td>
<td>xmlns:fmp=&quot;<a href="http://www.filemaker.com/fmpxmlresult">http://www.filemaker.com/fmpxmlresult</a>&quot;</td>
<td>fmp</td>
</tr>
<tr>
<td>FMPXMLLAYOUT grammar</td>
<td>xmlns:fml=&quot;<a href="http://www.filemaker.com/fmpxmllayout">http://www.filemaker.com/fmpxmllayout</a>&quot;</td>
<td>fml</td>
</tr>
<tr>
<td>For the query XML grammar</td>
<td>xmlns:fmq=&quot;<a href="http://www.filemaker.com/xml/query">http://www.filemaker.com/xml/query</a>&quot;</td>
<td>fmq</td>
</tr>
<tr>
<td>For the FileMaker XSLT extension functions</td>
<td>xmlns:fmxslt=&quot;xalan:///com.fmi.xslt.ExtensionFunctions&quot;</td>
<td>fmxslt</td>
</tr>
</tbody>
</table>

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 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 the Site Assistant and CDML Converter tools 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:

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

where:

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

  ```text
  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 “=” 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 25. 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.xsl:

  ```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.xsl 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.xsl:

  ```text
  ```

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

  ```text
  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:

   &-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 Administration Console. See the FileMaker Server Advanced Web Publishing Installation Guide.

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:

<table>
<thead>
<tr>
<th>Encoding</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>US-ASCII</td>
<td>The basic ASCII character set that is typically used for plain text English email.</td>
</tr>
<tr>
<td>ISO-8859-1</td>
<td>The Latin 1 character set that is typically used for roman character based web pages and email messages requiring upper ASCII characters.</td>
</tr>
<tr>
<td>ISO-8859-15</td>
<td>The Latin 9 character set, which is almost the same as the Latin 1 character set with the addition of the Euro € symbol.</td>
</tr>
<tr>
<td>ISO-2022-JP</td>
<td>The ISO Japanese encoding that is typically used for Japanese email messages.</td>
</tr>
<tr>
<td>Shift_JIS</td>
<td>The Japanese encoding that is typically used for Japanese web pages.</td>
</tr>
<tr>
<td>UTF-8</td>
<td>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.</td>
</tr>
</tbody>
</table>

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 Administration Console.

- You can also set email message encoding by using the fmxslt:send_email(String smtpFields, String body, String encoding) extension function. See “Sending email messages from the Web Publishing Engine” on page 61.
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:

```xml
<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 52.

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

```xml
<?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 `fmxslt:send_email()` that doesn’t require data from the database
- access information stored in a session if no database information is needed


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

For example:

```url
```
**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.xsl?-db=products&-lay=sales&-grammar=fmresultset
&-token.D100=Pendin&-findall
```

See “–token.[string] (Pass values between XSLT stylesheets) query parameter” on page 88.

**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 55.

**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 50.

The FileMaker XSLT extension functions have been designed so that you can use them within an XSLT stylesheets 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 48.

**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.

<table>
<thead>
<tr>
<th>FileMaker-specific XSLT parameter</th>
<th>For more information, see</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;xsl:param name=&quot;request-query&quot;/&gt;</code></td>
<td>“Accessing the query information in a request” in the next section.</td>
</tr>
<tr>
<td><code>&lt;xsl:param name=&quot;client-ip&quot;/&gt;</code></td>
<td>“Obtaining client information” on page 56.</td>
</tr>
<tr>
<td><code>&lt;xsl:param name=&quot;client-user-name&quot;/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;xsl:param name=&quot;client-password&quot;/&gt;</code></td>
<td></td>
</tr>
<tr>
<td><code>&lt;xsl:param name=&quot;authenticated-xml-base-uri&quot;/&gt;</code></td>
<td>“Using the authenticated base URI parameter” on page 56.</td>
</tr>
</tbody>
</table>
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:

```xml
<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:

```xml
<!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 50.

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

```
http://192.168.123.101/fmi/xsl/my_stylesheets.xsl?-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:

```xml
<query action="my_stylesheets.xsl" 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:

```xml
$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 \texttt{<xsl:template>} element.

You can use these parameters if you need to provide the web user’s credentials when programmatically loading additional password-protected XML documents while processing the original request. See “Loading additional documents” on page 57. The web user must provide the user name and password initially via the HTTP Basic Authentication dialog box. See “When web users use Custom Web Publishing to access a protected database” on page 17.

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 \texttt{<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:

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

Using the authenticated base URI parameter

The following 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 \texttt{<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:

```xml
<xsl:variable name="other-data" select="document(concat($xml-base-uri,'/fmi/xml/FMPXMLLAYOUT.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:

```xml
<xsl:variable name="other-data" select="document(concat($authenticated-xml-base-uri,'/fmi/xml/FMPXMLLAYOUT.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:

```xml
<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 will attempt 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:

```xml
<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 a database’s layout information in a stylesheet**

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

```xml
<xsl:variable name="layout" select="document(concat($xml-base-uri,'/fmi/xml/FMPXMLLAYOUT.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 value list named shirts that are defined in a layout in a FileMaker database. Here’s how you can use the document() function to load the layout information into a XSLT variable:

```xsl
<xsl:variable name="layout" select="document(concat($xml-base-uri,'/fmi/xml/FMPXMLLayout.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:FMPXMLLayout/fmi:VALUELISTS/fmi:VALUELIST[@NAME = 'shirts']/fmi:VALUE">
    <option>
      <xsl:attribute name="value"><xsl:value-of select="." /></xsl:attribute>
      <xsl:value-of select="." />
    </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 59.
- `fmxslt:set_header()`: See “Using the header functions” on page 62.
- `fmxslt:set_cookie()`: See “Using the cookie extension functions” on page 63.

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:

```xsl
<?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 `fmxslt: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 `fmxslt: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:

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

You should encode all links on a page as follows:

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

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

```html
<a href="my_stylesheet.xsl;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 `fmxslt: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.

<table>
<thead>
<tr>
<th>Session extension function</th>
<th>Data type returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmxslt:session_exists(String session-name)</code></td>
<td>boolean</td>
<td>Checks if a session with the specified name exists.</td>
</tr>
<tr>
<td><code>fmxslt:create_session(String session-name)</code></td>
<td>boolean</td>
<td>Creates a session with the specified session name and the default time-out, which is set via the Administration Console. See the <em>FileMaker Server Advanced Web Publishing Installation Guide</em>. <strong>Note</strong> This function requires the <code>&lt;?xslt-cwp-buffer?&gt;</code> processing instruction. See “Using content buffering” on page 58.</td>
</tr>
<tr>
<td><code>fmxslt:invalidate_session(String session-name)</code></td>
<td>boolean</td>
<td>Forces the session to time out immediately.</td>
</tr>
<tr>
<td><code>fmxslt:set_session_timeout(String session-name, Number timeout)</code></td>
<td>boolean</td>
<td>Sets the session timeout in minutes. The default timeout for sessions is set via the Administration Console.</td>
</tr>
<tr>
<td><code>fmxslt:session_encode_url(String url)</code></td>
<td>string</td>
<td>Encodes a URL with the session ID if the client does not support cookies; otherwise returns input URL.</td>
</tr>
</tbody>
</table>
Custom Web Publishing Guide

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

```xml
<xsl:variable name="session">
  <xsl:choose>
    <xsl:when test="not (fmxslt:session_exists(string($session-name)))">
      <xsl:value-of select="fmxslt:create_session(string($session-name))"/>
    </xsl:when>
    <xsl:otherwise>true</xsl:otherwise>
  </xsl:choose>
</xsl:variable>

<xsl:variable name="favorite-color" select="fmxslt: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 `fmxslt: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 Administration 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 the *FileMaker Server Advanced Web Publishing Installation Guide*.

### Session extension function

<table>
<thead>
<tr>
<th>Session extension function</th>
<th>Data type returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmxslt:set_session_object(String session-name, String name, Object value)</code></td>
<td>XSLT object (number, string, boolean, or node-set)</td>
<td>Stores an XSLT object (a number, string, boolean, or node-set) under a session, which can be later retrieved using the <code>fmxslt:get_session_object()</code> 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.</td>
</tr>
<tr>
<td><code>fmxslt:get_session_object(String session-name, String name)</code></td>
<td>XSLT object</td>
<td>Retrieves an XSLT object from the session.</td>
</tr>
<tr>
<td><code>fmxslt:remove_session_object(String session-name, String name)</code></td>
<td>XSLT object</td>
<td>Returns and then removes an XSLT object from the session.</td>
</tr>
</tbody>
</table>
**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 `fmxslt:send_email()` extension functions in an XSLT stylesheet. You can use these functions to send one or more separate messages. Because the `fmxslt: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.

<table>
<thead>
<tr>
<th>Email extension function</th>
<th>Data type returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmxslt:send_email(String smtpFields, String body)</code></td>
<td>boolean</td>
<td>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</td>
</tr>
<tr>
<td><code>fmxslt:send_email(String smtpFields, String body, String encoding)</code></td>
<td>boolean</td>
<td>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 52.</td>
</tr>
<tr>
<td><code>fmxslt:send_email(String smtpFields, String xsltFile, Node xml, boolean includeImages)</code></td>
<td>boolean</td>
<td>Sends an HTML-based email message using the encoding that is specified by encoding attribute of the <code>&lt;xsl:output&gt;</code> element in the stylesheet. If the encoding attribute is not included in the <code>&lt;xsl:output&gt;</code> element, the Web Publishing Engine’s default text encoding for email messages is used.</td>
</tr>
</tbody>
</table>

**Notes**

- In each of the three forms of the `fmxslt: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 “&amp;” 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 Administration Console. See the *FileMaker Server Advanced Web Publishing Installation Guide*.

The `fmxslt:send_email(String smtpFields, String xsltFile, Node xml, boolean includeImages)` 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 `fmxslt:send_email()` function.
- For the `includeImages` 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 `fmxslt:send_email(String smtpFields, String xsltFile, Node xml, boolean includeImages)` function inside an XPath statement, such as inside the `<xsl:if>` element:

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

For information about configuring the Web Publishing Engine to connect to a SMTP server, see the *FileMaker Server Advanced Web Publishing Installation Guide*.

**Using the header functions**

You can use the `fmxslt:get_header()` function to read information from the HTTP request and response headers, and the `fmxslt: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.

<table>
<thead>
<tr>
<th>Header extension function</th>
<th>Data type returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>fmxslt:get_header(String name)</code></td>
<td>string</td>
<td>Returns the specified header value</td>
</tr>
<tr>
<td><code>fmxslt:set_header(String name, String value)</code></td>
<td>void</td>
<td>Sets the specified header value</td>
</tr>
<tr>
<td><code>fmxslt:set_status_code(Number status-code)</code></td>
<td>void</td>
<td>Sets the HTTP status code</td>
</tr>
</tbody>
</table>
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 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.

```xml
<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.

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

Notes

- The `fmxslt:get_cookie()` and `fmxslt:get_cookies()` functions return a node-set in the following structure:
  ```xml
  <!ELEMENT cookie (#PCDATA)>
  <!ATTLIST cookie
    name CDATA #REQUIRED>
  ```
- 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.

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

```xml
<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.

<table>
<thead>
<tr>
<th>String manipulation function</th>
<th>Data type returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmxslt:break_encode(String value)</td>
<td>string</td>
<td>Returns an HTML break-encoded string. Characters such as “&amp;” (ampersand) are replaced with “&amp;”. New line characters such as line feeds and carriage returns are replaced with &lt;br/&gt;. This function works only if the disable-output-escaping attribute of the <a href="">xsl:value-of</a> and <a href="">xsl:text</a> elements is set to “yes” (disable-output-escaping=&quot;yes&quot;). <strong>Note</strong> 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.</td>
</tr>
<tr>
<td>fmxslt:html_encode(String value)</td>
<td>string</td>
<td>Returns an HTML-encoded string; characters such as “&amp;” (ampersand) are replaced with “&amp;”</td>
</tr>
<tr>
<td>fmxslt:url_encode(String value)</td>
<td>string</td>
<td>Returns a URL-encoded string. URL encoding is used to transmit characters over the Internet, particularly for URL’s. For example, the “&amp;” (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.</td>
</tr>
<tr>
<td>fmxslt:url_encode(String value, String encoding)</td>
<td>string</td>
<td>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 web site 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.</td>
</tr>
<tr>
<td>fmxslt:url_decode(String value)</td>
<td>string</td>
<td>Returns a URL-decoded string from a URL string that was previously encoded.</td>
</tr>
</tbody>
</table>
Comparing strings using Perl 5 regular expressions

You can use the `fmxslt: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:

```
fmxslt: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 `fmxslt: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="fmxslt:regex_contains($email, '^\w+[@\w.]*\@\w+([-\w.])*\.[a-z]{2,3}$')">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 97.

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:

```
fmxslt: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
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:

```
fmxsl:contains_checkbox_value($field_value, '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="$valuelists[@NAME = 'color_size']/fml:VALUE">
  <input type="checkbox">
    <xsl:attribute name="name">style</xsl:attribute>
    <xsl:attribute name="value"><xsl:value-of select="." /></xsl:attribute>
    <xsl:if test="fmxsl:contains_checkbox_value($field-value, ")">
      <xsl:attribute name="checked">checked</xsl:attribute>
    </xsl:if>
  </input><xsl:value-of select="." />
</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:

- [x] Red
- [ ] Blue
- [ ] Green
- [ ] Small
- [x] 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 these predefined date and time formats in the United States, which are defined by the defaults for the locale in the host computer where the Web Publishing Engine is installed:

- **Short date format is:** M/d/yy
- **Long date format is:** MMM d, yyyy
- **Short time format is:** h:mm a
- **Long time format is:** h:mm:ss a z
- **Short day format is:** EEE (in all locales). For example: “Sun”
- Long day format is: EEEE (in all locales) For example: “Sunday”

<table>
<thead>
<tr>
<th>Date, time, day extension function</th>
<th>Data type returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmxslt:get_date()</td>
<td>string</td>
<td>Returns the current date in the short format.</td>
</tr>
<tr>
<td>fmxslt:get_date(String format)</td>
<td>string</td>
<td>Returns the current date in the format you specify. Enter the string “short” or “long” for the format parameter.</td>
</tr>
<tr>
<td>fmxslt:get_time()</td>
<td>string</td>
<td>Returns the current time in the short format.</td>
</tr>
<tr>
<td>fmxslt:get_time(String format)</td>
<td>string</td>
<td>Returns the current time in the format you specify. Enter the string “short” or “long” for the format parameter.</td>
</tr>
<tr>
<td>fmxslt:get_day()</td>
<td>string</td>
<td>Returns the current day in the short format.</td>
</tr>
<tr>
<td>fmxslt:get_day(String format)</td>
<td>string</td>
<td>Returns the current day in the format you specify. Enter the string “short” or “long” for the format parameter.</td>
</tr>
<tr>
<td>fmxslt:get_short_date_format()</td>
<td>string</td>
<td>Returns the formatting string for short date format: “M/d/yy”.</td>
</tr>
<tr>
<td>fmxslt:get_long_date_format()</td>
<td>string</td>
<td>Returns the formatting string for long date format: “MMM d, yyyy”.</td>
</tr>
<tr>
<td>fmxslt:get_short_time_format()</td>
<td>string</td>
<td>Returns the formatting string for short time format: “h:mm a”.</td>
</tr>
<tr>
<td>fmxslt:get_long_time_format()</td>
<td>string</td>
<td>Returns the formatting string for long time format: “h:mm:ss a z”.</td>
</tr>
<tr>
<td>fmxslt:get_short_day_format()</td>
<td>string</td>
<td>Returns the formatting string for short day format: “EEE”.</td>
</tr>
<tr>
<td>fmxslt:get_long_day_format()</td>
<td>string</td>
<td>Returns the formatting string for long day format: “EEEE”.</td>
</tr>
<tr>
<td>fmxslt:compare_date(String date1, String date2)</td>
<td>number</td>
<td>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 short date format.</td>
</tr>
<tr>
<td>fmxslt:compare_time(String time1, String time2)</td>
<td>number</td>
<td>This function compares two time values. It returns a negative number if time1 is before time2. It returns a positive number if time1 is after time2. It returns a 0 if time1 is identical to time2. Both times must be specified in the short time format.</td>
</tr>
<tr>
<td>fmxslt:compare_day(String day1, String day2)</td>
<td>number</td>
<td>This function compares two day values. It returns a negative number if day1 is before day2. It returns a positive number if day1 is after day2. It returns a 0 if day1 is identical to day2. Both days must be specified in the short day format.</td>
</tr>
</tbody>
</table>
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.”

<table>
<thead>
<tr>
<th>Date, time, day extension function</th>
<th>Data type returned</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>fmxslt:get_datetime(String dateFormat)</td>
<td>string</td>
<td>Returns the current date and time using the date and time format strings.</td>
</tr>
<tr>
<td>fmxslt:convert_datetime(String oldFormat, String newFormat, String date)</td>
<td>string</td>
<td>Converts the specified date in the specified oldFormat into the string according to the specified newFormat. The oldFormat and newFormat strings must be specified using the date and time format strings.</td>
</tr>
<tr>
<td>fmxslt:compare_datetime(String dateFormat1, String dateFormat2, String date1, String date2)</td>
<td>number</td>
<td>This function compares date1 and date2 by decoding these dates according to respective date formats. 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. The dateFormat1 and dateFormat2 strings must be specified using the date and time format strings.</td>
</tr>
</tbody>
</table>

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):

<table>
<thead>
<tr>
<th>Letter</th>
<th>Date or Time Component</th>
<th>Presentation</th>
<th>Examples</th>
</tr>
</thead>
<tbody>
<tr>
<td>G</td>
<td>Era designator</td>
<td>Text</td>
<td>AD</td>
</tr>
<tr>
<td>y</td>
<td>Year</td>
<td>Year</td>
<td>1996; 96</td>
</tr>
<tr>
<td>M</td>
<td>Month in year</td>
<td>Month</td>
<td>July; Jul; 07</td>
</tr>
<tr>
<td>w</td>
<td>Week in year</td>
<td>Number</td>
<td>27</td>
</tr>
<tr>
<td>W</td>
<td>Week in month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>D</td>
<td>Day in year</td>
<td>Number</td>
<td>189</td>
</tr>
<tr>
<td>d</td>
<td>Day in month</td>
<td>Number</td>
<td>10</td>
</tr>
<tr>
<td>F</td>
<td>Day of week in month</td>
<td>Number</td>
<td>2</td>
</tr>
<tr>
<td>E</td>
<td>Day in week</td>
<td>Text</td>
<td>Tuesday; Tue</td>
</tr>
<tr>
<td>a</td>
<td>Am/pm marker</td>
<td>Text</td>
<td>PM</td>
</tr>
<tr>
<td>H</td>
<td>Hour in day (0-23)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>k</td>
<td>Hour in day (1-24)</td>
<td>Number</td>
<td>24</td>
</tr>
<tr>
<td>K</td>
<td>Hour in am/pm (0-11)</td>
<td>Number</td>
<td>0</td>
</tr>
<tr>
<td>h</td>
<td>Hour in am/pm (1-12)</td>
<td>Number</td>
<td>12</td>
</tr>
<tr>
<td>m</td>
<td>Minute in hour</td>
<td>Number</td>
<td>30</td>
</tr>
<tr>
<td>s</td>
<td>Second in minute</td>
<td>Number</td>
<td>55</td>
</tr>
<tr>
<td>S</td>
<td>Millisecond</td>
<td>Number</td>
<td>978</td>
</tr>
</tbody>
</table>
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` or `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.

<table>
<thead>
<tr>
<th>Date and Time Pattern</th>
<th>Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>&quot;yyyy.MM.dd G 'at' HH:mm:ss z&quot;</td>
<td>2001.07.04 AD at 12:08:56 PDT</td>
</tr>
<tr>
<td>&quot;EEE, MMM d, ' 'yy&quot;</td>
<td>Wed, Jul 4, '01</td>
</tr>
<tr>
<td>&quot;h:mm a&quot;</td>
<td>12:08 PM</td>
</tr>
<tr>
<td>&quot;hh 'o' 'clock' a, zzzz&quot;</td>
<td>12 o'clock PM, Pacific Daylight Time</td>
</tr>
<tr>
<td>&quot;K:mm a, z&quot;</td>
<td>0:08 PM, PDT</td>
</tr>
<tr>
<td>&quot;yyyyy.MMMMM.dd GGG hh:mm aaa&quot;</td>
<td>02001.July.04 AD 12:08 PM</td>
</tr>
<tr>
<td>&quot;EEE, d MMM yyyy HH:mm:ss Z&quot;</td>
<td>Wed, 4 Jul 2001 12:08:56-0700</td>
</tr>
<tr>
<td>&quot;yyMMddHHmssZ&quot;</td>
<td>010704120856-0700</td>
</tr>
</tbody>
</table>

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:

\[
\text{fmxslt:check_error_status()}
\]

When the \text{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 97.

**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 73.
Chapter 6

Testing and monitoring a site

Test your Custom Web Publishing site before deploying it. You can use log files to monitor the site during testing or after deployment.

Testing a Custom Web Publishing site

Before notifying users that your Custom Web Publishing site is available, it is important to 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 various 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 13 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 the web server, Web Publishing Engine, and FileMaker Server on one computer, 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 23, and “About the URL syntax for FileMaker XSLT stylesheets” on page 48.

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

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.

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’s web site 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 &lt;xsl:message&gt; element in XSLT stylesheets. Whenever web users access your XSLT stylesheet, information you’ve included within a &lt;xsl:message&gt; element is recorded in the application log file. See chapter 5, “Developing FileMaker XSLT stylesheets.”

The application log is named pe_application_log.txt and is located in the Logs folder in the Web Publishing folder in the FileMaker Server 7 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:

- Error Logging
- Script Logging
- User Logging

All three of these Logging options are enabled by default. For information on setting these options via the Administration Console, see the FileMaker Server Advanced Web Publishing Installation Guide.

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, consider archiving 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 is a record of any errors with its operation. This file is called web_server_module_log.txt and is located in the Logs folder in the Web Publishing folder in the FileMaker Server 7 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 Web Publishing folder in the FileMaker Server 7 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 via the Administration Console, see the *FileMaker Server Advanced Web Publishing Installation Guide*. 
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:

<table>
<thead>
<tr>
<th>Query command names</th>
<th>Query parameter names</th>
</tr>
</thead>
<tbody>
<tr>
<td>–dbnames (See page 79.)</td>
<td>–db (See page 82.)</td>
</tr>
<tr>
<td>–delete (See page 79.)</td>
<td>–encoding (XSLT only) (See page 82.)</td>
</tr>
<tr>
<td>–dup (See page 79.)</td>
<td>–field (See page 82.)</td>
</tr>
<tr>
<td>–edit (See page 80.)</td>
<td>fieldname (See page 82.)</td>
</tr>
<tr>
<td>–find, –findall, –findany (See page 80.)</td>
<td>fieldname.op (See page 83.)</td>
</tr>
<tr>
<td>–layoutnames (See page 80.)</td>
<td>–grammar (XSLT only) (See page 84.)</td>
</tr>
<tr>
<td>–new (See page 81.)</td>
<td>–lay (See page 84.)</td>
</tr>
<tr>
<td>–process (XSLT only) (See page 81.)</td>
<td>–lay.response (See page 84.)</td>
</tr>
<tr>
<td>–scriptnames (See page 81.)</td>
<td>–lop (See page 84.)</td>
</tr>
<tr>
<td>–view (See page 81.)</td>
<td>–max (See page 85.)</td>
</tr>
<tr>
<td></td>
<td>–modid (See page 85.)</td>
</tr>
<tr>
<td></td>
<td>–recid (See page 85.)</td>
</tr>
<tr>
<td></td>
<td>–script (See page 85.)</td>
</tr>
<tr>
<td></td>
<td>–script.prefind (See page 86.)</td>
</tr>
<tr>
<td></td>
<td>–script.presort (See page 86.)</td>
</tr>
<tr>
<td></td>
<td>–skip (See page 86.)</td>
</tr>
<tr>
<td></td>
<td>–sortfield.[1-9] (See page 86.)</td>
</tr>
<tr>
<td></td>
<td>–sortorder.[1-9] (See page 87.)</td>
</tr>
<tr>
<td></td>
<td>–styletype (See page 87.)</td>
</tr>
<tr>
<td></td>
<td>–stylehref (See page 88.)</td>
</tr>
<tr>
<td></td>
<td>–token.[string] (XSLT only) (See page 88.)</td>
</tr>
</tbody>
</table>

**Important** The –lay parameter for specifying a layout is now required with all query commands except –dbnames, –layoutnames, and –scriptnames.
**Obsolete request names and parameters**

The following request name and parameters are no longer supported:

<table>
<thead>
<tr>
<th>Obsolete request names</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>–dbopen</td>
<td>Opened a database that was in the Web folder with Remote Administration enabled.</td>
</tr>
<tr>
<td>–dbclose</td>
<td>Closed a database that’s in the Web folder with Remote Administration enabled.</td>
</tr>
<tr>
<td>–img</td>
<td>Retrieved the specified image. Use a container URL instead. See “About the URL syntax for FileMaker container objects in XML solutions” on page 24, and “About the URL syntax for FileMaker container objects in XSLT solutions” on page 48.</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Obsolete parameter names</th>
<th>Comment</th>
</tr>
</thead>
<tbody>
<tr>
<td>–format</td>
<td>Specified a format file for CDML or XML grammar. CDML is no longer supported. See appendix C, “Converting CDML solutions to FileMaker XSLT.”</td>
</tr>
<tr>
<td>–fmtfield</td>
<td>Obtained format file information from a field in the database.</td>
</tr>
</tbody>
</table>

**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 –dbname 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 33.
- 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:

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

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

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**

The FileMaker Server Advanced CD 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. You can open the FileMaker Query Strings Reference in the Custom Web Publishing Reference folder on the FileMaker Server Advanced CD.

**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:

```plaintext
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.

**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 fmresultset grammar” on page 27.

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

**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 77. 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&-edit

**Note** When you use the –edit command to set the value of a global field, you don’t need to use the –recid parameter if you are only setting the global field value with that request.

**Important** If you are using a global field in an XSLT stylesheet, you must use the Administration 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 the *FileMaker Server Advanced Web Publishing Installation Guide*.

---

**Using the query commands**

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 78.

Example:
To edit a record:

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

Submits a search request using defined criteria

Required query parameters: –db, –lay


Examples:
To find a record by field name:
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family&Country=USA&-find

To find a record by record ID:
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family&-recid=427&-find

To find all records in the database:
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family&-findall

To find a random record:
http://192.168.123.101/fmi/xml/fmresultset.xml?-db=employees&-lay=family&-findany

–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
Valid names used in query strings

–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 77.

**Example:**
To add a new record:

–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:**

See “Processing XSLT requests that do not query FileMaker Server” on page 53.

–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
Using the query parameters

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 49.

–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

–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.


**Optional with:** all query commands in an XSLT request

**Example:**
http://192.168.123.101/fmi/xsl/my_template/my_stylesheet.xsl?-db=employees&-lay=departments&-grammar=fmresultset &-encoding=Shift_JIS&-findall

See “Setting text encoding for requests” on page 52.

–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 24, and “About the URL syntax for FileMaker container objects in XSLT solutions” on page 48.

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. See “About the syntax for a fully qualified field name” on page 77.

**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.
Valid names used in query strings

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

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:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>FileMaker Pro equivalent operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>eq</td>
<td>=word</td>
</tr>
<tr>
<td>cn</td>
<td><em>word</em></td>
</tr>
<tr>
<td>bw</td>
<td>word*</td>
</tr>
<tr>
<td>ew</td>
<td>*word</td>
</tr>
<tr>
<td>gt</td>
<td>&gt; word</td>
</tr>
<tr>
<td>gte</td>
<td>&gt;= word</td>
</tr>
<tr>
<td>lt</td>
<td>&lt; word</td>
</tr>
<tr>
<td>lte</td>
<td>&lt;= word</td>
</tr>
<tr>
<td>neq</td>
<td>omit, word</td>
</tr>
</tbody>
</table>

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.xsl?-grammar=fmresultset&-db=mydatabase

&-lay=mylayout&-findall

See “Specifying an XML grammar for a FileMaker XSLT stylesheet” on page 50.

-**lay (Layout) query parameter**

Specifies the database layout you want to use

**Value is:** Name of the layout

**Required with:** All query commands except –process, –dbnames, –layoutnames, and –scriptnames

**Example:**

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

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

Specifies that FileMaker Server should use the layout specified by the –lay parameter when processing a request, and switch to the layout specified by the –lay.response parameter when processing the XML response. If you don’t include the –lay.response parameter, FileMaker Server uses the layout specified by the –lay parameter when processing both the request and the response. You can use the –lay.response parameter for either XML requests or in XSLT stylesheet requests.

**Value is:** Name of the layout

**Optional with:** All query commands except –process, –dbnames, –layoutnames, and –scriptnames

**Example:**

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

&-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

**Example:**

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

&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 or –findall 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

-recid (Record ID) query parameter

Specifies the record you want processed. Used mainly by the –edit, and –delete query commands.

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 command

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

-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 35.

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.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.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

–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:

<table>
<thead>
<tr>
<th>Keyword</th>
<th>FileMaker Pro Equivalent Operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>ascend</td>
<td>Sort a to z, –10 to 10</td>
</tr>
<tr>
<td>descend</td>
<td>Sort z to a, 10 to –10</td>
</tr>
<tr>
<td><code>&lt;value-list-name&gt;</code></td>
<td>Sort using the specified value list associated with the field on the layout</td>
</tr>
</tbody>
</table>

**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:

```
-<sortfield>.<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.

**–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/xsl)—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 35. 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
-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.xsl)—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 .xsl. See “Using server-side and client-side processing of stylesheets” on page 35. This parameter is used in conjunction with the –styletype parameter.

Optional with: All query commands

Requires: –styletype parameter

Example (assumes mystylesheet.xsl 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/xsl
&-stylehref=/mystylesheet.xsl&-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.xsl?-db=employees&-lay=departments&-grammar=fmresultset
&-token.D100=Active&-findall

See “Using tokens to pass information between stylesheets” on page 54.
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. For information, 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 95.

- **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 97.

### 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 FMPDSORESULT 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
<?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>
</fmresultset>
```

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

```xml
<?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>
</fmpxmlresult>
```

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.

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Unknown error</td>
</tr>
<tr>
<td>0</td>
<td>No error</td>
</tr>
<tr>
<td>1</td>
<td>User canceled action</td>
</tr>
<tr>
<td>2</td>
<td>Memory error</td>
</tr>
<tr>
<td>3</td>
<td>Command is unavailable (for example, wrong operating system, wrong mode, etc.)</td>
</tr>
<tr>
<td>4</td>
<td>Command is unknown</td>
</tr>
<tr>
<td>5</td>
<td>Command is invalid (for example, a Set Field script step does not have a calculation specified)</td>
</tr>
<tr>
<td>Error Number</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>6</td>
<td>File is read-only</td>
</tr>
<tr>
<td>7</td>
<td>Running out of memory</td>
</tr>
<tr>
<td>8</td>
<td>Empty result</td>
</tr>
<tr>
<td>9</td>
<td>Insufficient privileges</td>
</tr>
<tr>
<td>10</td>
<td>Requested data is missing</td>
</tr>
<tr>
<td>11</td>
<td>Name is not valid</td>
</tr>
<tr>
<td>12</td>
<td>Name already exists</td>
</tr>
<tr>
<td>13</td>
<td>File or object is in use</td>
</tr>
<tr>
<td>14</td>
<td>Out of range</td>
</tr>
<tr>
<td>15</td>
<td>Can’t divide by zero</td>
</tr>
<tr>
<td>16</td>
<td>Operation failed, request retry (for example, a user query)</td>
</tr>
<tr>
<td>17</td>
<td>Attempt to convert foreign character set to UTF-16 failed</td>
</tr>
<tr>
<td>18</td>
<td>Client must provide account information to proceed</td>
</tr>
<tr>
<td>19</td>
<td>String contains characters other than A-Z, a-z, 0-9 (ASCII)</td>
</tr>
<tr>
<td>100</td>
<td>File is missing</td>
</tr>
<tr>
<td>101</td>
<td>Record is missing</td>
</tr>
<tr>
<td>102</td>
<td>Field is missing</td>
</tr>
<tr>
<td>103</td>
<td>Relationship is missing</td>
</tr>
<tr>
<td>104</td>
<td>Script is missing</td>
</tr>
<tr>
<td>105</td>
<td>Layout is missing</td>
</tr>
<tr>
<td>106</td>
<td>Table is missing</td>
</tr>
<tr>
<td>107</td>
<td>Index is missing</td>
</tr>
<tr>
<td>108</td>
<td>Value list is missing</td>
</tr>
<tr>
<td>109</td>
<td>Privilege set is missing</td>
</tr>
<tr>
<td>110</td>
<td>Related tables are missing</td>
</tr>
<tr>
<td>111</td>
<td>Field repetition is invalid</td>
</tr>
<tr>
<td>112</td>
<td>Window is missing</td>
</tr>
<tr>
<td>113</td>
<td>Function is missing</td>
</tr>
<tr>
<td>114</td>
<td>File reference is missing</td>
</tr>
<tr>
<td>130</td>
<td>Files are damaged or missing and must be reinstalled</td>
</tr>
<tr>
<td>131</td>
<td>Language pack files are missing (such as template files)</td>
</tr>
<tr>
<td>200</td>
<td>Record access is denied</td>
</tr>
<tr>
<td>201</td>
<td>Field cannot be modified</td>
</tr>
<tr>
<td>202</td>
<td>Field access is denied</td>
</tr>
<tr>
<td>203</td>
<td>No records in file to print, or password doesn’t allow print access</td>
</tr>
<tr>
<td>Error Number</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-------------</td>
</tr>
<tr>
<td>204</td>
<td>No access to field(s) in sort order</td>
</tr>
<tr>
<td>205</td>
<td>User does not have access privileges to create new records; import will overwrite existing data</td>
</tr>
<tr>
<td>206</td>
<td>User does not have password change privileges, or file is not modifiable</td>
</tr>
<tr>
<td>207</td>
<td>User does not have sufficient privileges to change database schema, or file is not modifiable</td>
</tr>
<tr>
<td>208</td>
<td>Password does not contain enough characters</td>
</tr>
<tr>
<td>209</td>
<td>New password must be different from existing one</td>
</tr>
<tr>
<td>210</td>
<td>User account is inactive</td>
</tr>
<tr>
<td>211</td>
<td>Password has expired</td>
</tr>
<tr>
<td>212</td>
<td>Invalid user account and/or password. Please try again</td>
</tr>
<tr>
<td>213</td>
<td>User account and/or password does not exist</td>
</tr>
<tr>
<td>214</td>
<td>Too many login attempts</td>
</tr>
<tr>
<td>215</td>
<td>Administrator privileges cannot be duplicated</td>
</tr>
<tr>
<td>216</td>
<td>Guest account cannot be duplicated</td>
</tr>
<tr>
<td>217</td>
<td>User does not have sufficient privileges to modify administrator account</td>
</tr>
<tr>
<td>300</td>
<td>File is locked or in use</td>
</tr>
<tr>
<td>301</td>
<td>Record is in use by another user</td>
</tr>
<tr>
<td>302</td>
<td>Table is in use by another user</td>
</tr>
<tr>
<td>303</td>
<td>Database schema is in use by another user</td>
</tr>
<tr>
<td>304</td>
<td>Layout is in use by another user</td>
</tr>
<tr>
<td>306</td>
<td>Record modification ID does not match</td>
</tr>
<tr>
<td>400</td>
<td>Find criteria are empty</td>
</tr>
<tr>
<td>401</td>
<td>No records match the request</td>
</tr>
<tr>
<td>402</td>
<td>Selected field is not a match field for a lookup</td>
</tr>
<tr>
<td>403</td>
<td>Exceeding maximum record limit for trial version of FileMaker Pro</td>
</tr>
<tr>
<td>404</td>
<td>Sort order is invalid</td>
</tr>
<tr>
<td>405</td>
<td>Number of records specified exceeds number of records that can be omitted</td>
</tr>
<tr>
<td>406</td>
<td>Replace/Reserialize criteria are invalid</td>
</tr>
<tr>
<td>407</td>
<td>One or both match fields are missing (invalid relationship)</td>
</tr>
<tr>
<td>408</td>
<td>Specified field has inappropriate data type for this operation</td>
</tr>
<tr>
<td>409</td>
<td>Import order is invalid</td>
</tr>
<tr>
<td>410</td>
<td>Export order is invalid</td>
</tr>
<tr>
<td>412</td>
<td>Wrong version of FileMaker Pro used to recover file</td>
</tr>
<tr>
<td>413</td>
<td>Specified field has inappropriate field type</td>
</tr>
<tr>
<td>414</td>
<td>Layout cannot display the result</td>
</tr>
<tr>
<td>500</td>
<td>Date value does not meet validation entry options</td>
</tr>
<tr>
<td>Error Number</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>501</td>
<td>Time value does not meet validation entry options</td>
</tr>
<tr>
<td>502</td>
<td>Number value does not meet validation entry options</td>
</tr>
<tr>
<td>503</td>
<td>Value in field is not within the range specified in validation entry options</td>
</tr>
<tr>
<td>504</td>
<td>Value in field is not unique as required in validation entry options</td>
</tr>
<tr>
<td>505</td>
<td>Value in field is not an existing value in the database file as required in validation entry options</td>
</tr>
<tr>
<td>506</td>
<td>Value in field is not listed on the value list specified in validation entry option</td>
</tr>
<tr>
<td>507</td>
<td>Value in field failed calculation test of validation entry option</td>
</tr>
<tr>
<td>508</td>
<td>Invalid value entered in Find mode</td>
</tr>
<tr>
<td>509</td>
<td>Field requires a valid value</td>
</tr>
<tr>
<td>510</td>
<td>Related value is empty or unavailable</td>
</tr>
<tr>
<td>511</td>
<td>Value in field exceeds maximum number of allowed characters</td>
</tr>
<tr>
<td>600</td>
<td>Print error has occurred</td>
</tr>
<tr>
<td>601</td>
<td>Combined header and footer exceed one page</td>
</tr>
<tr>
<td>602</td>
<td>Body doesn’t fit on a page for current column setup</td>
</tr>
<tr>
<td>603</td>
<td>Print connection lost</td>
</tr>
<tr>
<td>700</td>
<td>File is of the wrong file type for import</td>
</tr>
<tr>
<td>706</td>
<td>EPSF file has no preview image</td>
</tr>
<tr>
<td>707</td>
<td>Graphic translator cannot be found</td>
</tr>
<tr>
<td>708</td>
<td>Can’t import the file or need color monitor support to import file</td>
</tr>
<tr>
<td>709</td>
<td>QuickTime movie import failed</td>
</tr>
<tr>
<td>710</td>
<td>Unable to update QuickTime file reference because the database file is read-only</td>
</tr>
<tr>
<td>711</td>
<td>Import translator cannot be found</td>
</tr>
<tr>
<td>714</td>
<td>Password privileges do not allow the operation</td>
</tr>
<tr>
<td>715</td>
<td>Specified Excel worksheet or named range is missing</td>
</tr>
<tr>
<td>716</td>
<td>A SQL query using DELETE, INSERT, or UPDATE is not allowed for ODBC import</td>
</tr>
<tr>
<td>717</td>
<td>There is not enough XML/XSL information to proceed with the import or export</td>
</tr>
<tr>
<td>718</td>
<td>Error in parsing XML file (from Xerces)</td>
</tr>
<tr>
<td>719</td>
<td>Error in transforming XML using XSL (from Xalan)</td>
</tr>
<tr>
<td>720</td>
<td>Error when exporting; intended format does not support repeating fields</td>
</tr>
<tr>
<td>721</td>
<td>Unknown error occurred in the parser or the transformer</td>
</tr>
<tr>
<td>722</td>
<td>Cannot import data into a file that has no fields</td>
</tr>
<tr>
<td>723</td>
<td>You do not have permission to add records to or modify records in the target table</td>
</tr>
<tr>
<td>724</td>
<td>You do not have permission to add records to the target table</td>
</tr>
<tr>
<td>725</td>
<td>You do not have permission to modify records in the target table</td>
</tr>
<tr>
<td>726</td>
<td>There are more records in the import file than in the target table. Not all records were imported</td>
</tr>
<tr>
<td>Error Number</td>
<td>Description</td>
</tr>
<tr>
<td>--------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>727</td>
<td>There are more records in the target table than in the import file. Not all records were updated</td>
</tr>
<tr>
<td>729</td>
<td>Errors occurred during import. Records could not be imported</td>
</tr>
<tr>
<td>730</td>
<td>Unsupported Excel version. (Convert file to Excel 7.0 (Excel 95), Excel 97, 2000, or XP format and try again)</td>
</tr>
<tr>
<td>731</td>
<td>The file you are importing from contains no data</td>
</tr>
<tr>
<td>732</td>
<td>This file cannot be inserted because it contains other files</td>
</tr>
<tr>
<td>733</td>
<td>A table cannot be imported into itself</td>
</tr>
<tr>
<td>734</td>
<td>This file type cannot be displayed as a picture</td>
</tr>
<tr>
<td>735</td>
<td>This file type cannot be displayed as a picture. It will be inserted and displayed as a file</td>
</tr>
<tr>
<td>800</td>
<td>Unable to create file on disk</td>
</tr>
<tr>
<td>801</td>
<td>Unable to create temporary file on System disk</td>
</tr>
<tr>
<td>802</td>
<td>Unable to open file</td>
</tr>
<tr>
<td>803</td>
<td>File is single user or host cannot be found</td>
</tr>
<tr>
<td>804</td>
<td>File cannot be opened as read-only in its current state</td>
</tr>
<tr>
<td>805</td>
<td>File is damaged; use Recover command</td>
</tr>
<tr>
<td>806</td>
<td>File cannot be opened with this version of FileMaker Pro</td>
</tr>
<tr>
<td>807</td>
<td>File is not a FileMaker Pro file or is severely damaged</td>
</tr>
<tr>
<td>808</td>
<td>Cannot open file because access privileges are damaged</td>
</tr>
<tr>
<td>809</td>
<td>Disk/volume is full</td>
</tr>
<tr>
<td>810</td>
<td>Disk/volume is locked</td>
</tr>
<tr>
<td>811</td>
<td>Temporary file cannot be opened as FileMaker Pro file</td>
</tr>
<tr>
<td>813</td>
<td>Record Synchronization error on network</td>
</tr>
<tr>
<td>814</td>
<td>File(s) cannot be opened because maximum number is open</td>
</tr>
<tr>
<td>815</td>
<td>Couldn’t open lookup file</td>
</tr>
<tr>
<td>816</td>
<td>Unable to convert file</td>
</tr>
<tr>
<td>817</td>
<td>Unable to open file because it does not belong to this solution</td>
</tr>
<tr>
<td>819</td>
<td>Cannot save a local copy of a remote file</td>
</tr>
<tr>
<td>820</td>
<td>File is in the process of being closed</td>
</tr>
<tr>
<td>821</td>
<td>Host forced a disconnect</td>
</tr>
<tr>
<td>822</td>
<td>FMI files not found; reinstall missing files</td>
</tr>
<tr>
<td>823</td>
<td>Cannot set file to single-user, guests are connected</td>
</tr>
<tr>
<td>824</td>
<td>File is damaged or not a FileMaker file</td>
</tr>
<tr>
<td>900</td>
<td>General spelling engine error</td>
</tr>
<tr>
<td>901</td>
<td>Main spelling dictionary not installed</td>
</tr>
<tr>
<td>902</td>
<td>Could not launch the Help system</td>
</tr>
<tr>
<td>903</td>
<td>Command cannot be used in a shared file</td>
</tr>
<tr>
<td>Error Number</td>
<td>Description</td>
</tr>
<tr>
<td>------------</td>
<td>-----------------------------------------------------------------------------</td>
</tr>
<tr>
<td>904</td>
<td>Command can only be used in a file hosted under FileMaker Server</td>
</tr>
<tr>
<td>905</td>
<td>No active field selected; command can only be used if there is an active field</td>
</tr>
<tr>
<td>920</td>
<td>Can’t initialize the spelling engine</td>
</tr>
<tr>
<td>921</td>
<td>User dictionary cannot be loaded for editing</td>
</tr>
<tr>
<td>922</td>
<td>User dictionary cannot be found</td>
</tr>
<tr>
<td>923</td>
<td>User dictionary is read-only</td>
</tr>
<tr>
<td>951</td>
<td>An unexpected error occurred</td>
</tr>
<tr>
<td>954</td>
<td>Unsupported XML grammar</td>
</tr>
<tr>
<td>955</td>
<td>No database name</td>
</tr>
<tr>
<td>956</td>
<td>Maximum number of database sessions exceeded</td>
</tr>
<tr>
<td>957</td>
<td>Conflicting commands</td>
</tr>
<tr>
<td>958</td>
<td>Parameter missing in query</td>
</tr>
<tr>
<td>1200</td>
<td>Generic calculation error</td>
</tr>
<tr>
<td>1201</td>
<td>Too few parameters in the function</td>
</tr>
<tr>
<td>1202</td>
<td>Too many parameters in the function</td>
</tr>
<tr>
<td>1203</td>
<td>Unexpected end of calculation</td>
</tr>
<tr>
<td>1204</td>
<td>Number, text constant, field name or “(“ expected</td>
</tr>
<tr>
<td>1205</td>
<td>Comment is not terminated with &quot;*/&quot;</td>
</tr>
<tr>
<td>1206</td>
<td>Text constant must end with a quotation mark</td>
</tr>
<tr>
<td>1207</td>
<td>Unbalanced parenthesis</td>
</tr>
<tr>
<td>1208</td>
<td>Operator missing, function not found or “(“ not expected</td>
</tr>
<tr>
<td>1209</td>
<td>Name (such as field name or layout name) is missing</td>
</tr>
<tr>
<td>1210</td>
<td>Plug-in function has already been registered</td>
</tr>
<tr>
<td>1211</td>
<td>List usage is not allowed in this function</td>
</tr>
<tr>
<td>1212</td>
<td>An operator (for example, +, -, *) is expected here</td>
</tr>
<tr>
<td>1213</td>
<td>This variable has already been defined in the Let function</td>
</tr>
<tr>
<td>1214</td>
<td>AVERAGE, COUNT, EXTEND, GETREPETITION, MAX, MIN, NPV, STDEV, SUM and GETSUMMARY: expression found where a field alone is needed</td>
</tr>
<tr>
<td>1215</td>
<td>This parameter is an invalid Get function parameter</td>
</tr>
<tr>
<td>1216</td>
<td>Only Summary fields allowed as first argument in GETSUMMARY</td>
</tr>
<tr>
<td>1217</td>
<td>Break field is invalid</td>
</tr>
<tr>
<td>1218</td>
<td>Cannot evaluate the number</td>
</tr>
<tr>
<td>1219</td>
<td>A field cannot be used in its own formula</td>
</tr>
<tr>
<td>1220</td>
<td>Field type must be normal or calculated</td>
</tr>
<tr>
<td>1221</td>
<td>Data type must be number, date, time, or timestamp</td>
</tr>
<tr>
<td>1222</td>
<td>Calculation cannot be stored</td>
</tr>
</tbody>
</table>
Error codes 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:

<table>
<thead>
<tr>
<th>Error Number</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>1223</td>
<td>The function referred to does not exist</td>
</tr>
<tr>
<td>1400</td>
<td>ODBC driver initialization failed; make sure the ODBC drivers are properly installed.</td>
</tr>
<tr>
<td>1401</td>
<td>Failed to allocate environment (ODBC)</td>
</tr>
<tr>
<td>1402</td>
<td>Failed to free environment (ODBC)</td>
</tr>
<tr>
<td>1403</td>
<td>Failed to disconnect (ODBC)</td>
</tr>
<tr>
<td>1404</td>
<td>Failed to allocate connection (ODBC)</td>
</tr>
<tr>
<td>1405</td>
<td>Failed to free connection (ODBC)</td>
</tr>
<tr>
<td>1406</td>
<td>Failed check for SQL API (ODBC)</td>
</tr>
<tr>
<td>1407</td>
<td>Failed to allocate statement (ODBC)</td>
</tr>
<tr>
<td>1408</td>
<td>Extended error (ODBC)</td>
</tr>
</tbody>
</table>
Here is a partial list of the Web Publishing Engine error code values:

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

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 the FileMaker Server Advanced Web Publishing Installation Guide.
**Error code numbers for the FileMaker XSLT extension functions**

The `fmxslt:check_error_status()` extension function (see “Checking the error status of extension functions” on page 70), returns one of the errors in the following table:

<table>
<thead>
<tr>
<th>Error code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-1</td>
<td>Unknown error</td>
</tr>
<tr>
<td>0</td>
<td>No error</td>
</tr>
</tbody>
</table>

### General Errors

<table>
<thead>
<tr>
<th>Error code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10000</td>
<td>Invalid header name</td>
</tr>
<tr>
<td>10001</td>
<td>Invalid HTTP status code</td>
</tr>
</tbody>
</table>

### Session Errors

<table>
<thead>
<tr>
<th>Error code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10100</td>
<td>Unknown session error</td>
</tr>
<tr>
<td>10101</td>
<td>Requested session name is already used</td>
</tr>
<tr>
<td>10102</td>
<td>Session could not be accessed - maybe it does not exist</td>
</tr>
<tr>
<td>10103</td>
<td>Session has timed out</td>
</tr>
<tr>
<td>10104</td>
<td>Specified session object does not exist</td>
</tr>
</tbody>
</table>

### Messaging Errors

<table>
<thead>
<tr>
<th>Error code value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>10200</td>
<td>Unknown messaging error</td>
</tr>
<tr>
<td>10201</td>
<td>Message formatting error</td>
</tr>
<tr>
<td>10202</td>
<td>Message SMTP fields error</td>
</tr>
<tr>
<td>10203</td>
<td>Message “To Field” error</td>
</tr>
<tr>
<td>10204</td>
<td>Message “From Field” error</td>
</tr>
<tr>
<td>10205</td>
<td>Message “CC Field” error</td>
</tr>
<tr>
<td>10206</td>
<td>Message “BCC Field” error</td>
</tr>
<tr>
<td>10207</td>
<td>Message “Subject Field” error</td>
</tr>
<tr>
<td>10208</td>
<td>Message “Reply-To Field” error</td>
</tr>
<tr>
<td>10209</td>
<td>Message body error</td>
</tr>
<tr>
<td>10210</td>
<td>Recursive mail error - attempted to call send_email() inside an email XSLT stylesheet</td>
</tr>
<tr>
<td>10211</td>
<td>SMTP authentication error - either login failed or wrong type of authentication provided</td>
</tr>
<tr>
<td>10212</td>
<td>Invalid function usage - attempted to call set_header(), set_status_code() or set_cookie() inside an email XSLT stylesheet</td>
</tr>
<tr>
<td>Error code value</td>
<td>Description</td>
</tr>
<tr>
<td>------------------</td>
<td>----------------------------------------------</td>
</tr>
<tr>
<td>10300</td>
<td>Unknown formatting error</td>
</tr>
<tr>
<td>10301</td>
<td>Invalid date time format</td>
</tr>
<tr>
<td>10302</td>
<td>Invalid date format</td>
</tr>
<tr>
<td>10303</td>
<td>Invalid time format</td>
</tr>
<tr>
<td>10304</td>
<td>Invalid day format</td>
</tr>
<tr>
<td>10305</td>
<td>Improperly formatted date time string</td>
</tr>
<tr>
<td>10306</td>
<td>Improperly formatted date string</td>
</tr>
<tr>
<td>10307</td>
<td>Improperly formatted time string</td>
</tr>
<tr>
<td>10308</td>
<td>Improperly formatted day string</td>
</tr>
<tr>
<td>10309</td>
<td>Unsupported text encoding</td>
</tr>
<tr>
<td>10310</td>
<td>Invalid URL encoding</td>
</tr>
<tr>
<td>10311</td>
<td>Regular expression pattern error</td>
</tr>
</tbody>
</table>
Appendix C

Converting CDML solutions to FileMaker XSLT

This appendix explains the results of using the FileMaker CDML Converter to convert CDML format files to FileMaker XSLT stylesheets. For information on using the FileMaker CDML Converter, see “Using the FileMaker CDML Converter” on page 42.

Note In this appendix and in the generated comments in converted XSLT stylesheets, the term “XSLT-CWP” refers to FileMaker Custom Web Publishing with XSLT.

About the process of converting CDML solutions to FileMaker XSLT solutions

The CDML Converter converts, renames, or copies the CDML solution files in the source folder as follows:

- CDML format files that contain CDML tags or query parameters are converted to XSLT stylesheets and then saved in the destination folder.
- The filename extensions of HTML files, regardless of whether they contain CDML tags or not, are renamed to .xsl, and then the files are copied to the destination folder. For example, myfile.html is renamed to myfile.xsl.
- All upper case filenames are changed to lower case because all converted file references use lower case and XSLT is case sensitive.
- Non-HTML files that do not contain CDML tags or query parameters are copied without changes to the destination folder.
- Any folders within the source folder hierarchy are automatically created in the destination folder.

To convert each CDML format file to an XSLT stylesheet, the CDML Converter:

- removes the <!DOCTYPE> tag if it is included.
- maps all CDML boolean expressions to XPath expressions.
- in all occurrences of the –format CDML tag, converts all filename extensions to .xsl in filename references.
- maps all CDML replacement tags and intratag parameters to XSLT-CWP statements.
- inserts the <?xslt-cwp-query params="–grammar=fmresultset"?> processing instruction in all converted stylesheets to specify the fmresultset grammar. See “Using statically defined query commands and query parameters” on page 51.
- inserts a value for the encoding attribute in the <?xml ... ?> processing instruction and the <xsl:output> element that matches the text encoding option you selected in the CDML Converter during the conversion.
- inserts a value of “xml” for the method attribute of the <xsl:output> element if the input file begins with a <?xml ... ?> processing instruction. If the input file is an HTML file, “html” is inserted as the value of the method attribute; otherwise, “text” is inserted.
For all Standard Generalized Markup Language (SGML) input files, including XML and HTML files, the CDML Converter inserts line breaks where appropriate and indents nested elements. The CDML Converter also converts the files into well-formed XML documents by doing the following:

- inserts the following declarations for the namespaces, XSLT parameters, and variables that are used in a stylesheet. The declarations are inserted before and after the converted XSLT-CWP statements:

```xml
<?xml version="1.0" encoding="ISO-8859-1"?>
<xsl:stylesheet version="1.0"
    xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:fmr="http://www.filemaker.com/xml/fmresultset"
    xmlns:fml="http://www.filemaker.com/fmpxmllayout"
    xmlns:fmq="http://www.filemaker.com/xml/query"
    xmlns:fmsxlt="xalan://com.fmi.xslt.ExtensionFunctions"
    xmlns:xalan="http://xml.apache.org/xalan"
    exclude-result-prefixes="xsl fmr fml fmq fmsxlt xalan">
    <xsl:param name="authenticated-xml-base-uri"/>
    <xsl:param name="request-query"/>
    <xsl:variable name="layout" select="document(concat($authenticated-xml-base-uri,
        '/fmi/xml/FMPXMLLAYOUT.xml?','-db=', /fmr:fmresultset/fmr:database/@name,'&-
        lay=', /fmr:fmresultset/fmr:database/@layout, '&-view='))"/>
    <xsl:variable name="current-find"><xsl:call-template name="get-current-find"/></xsl:variable>
    <xsl:variable name="current-sort"><xsl:call-template name="get-current-sort"/></xsl:variable>
    <xsl:variable name="current-action"><xsl:call-template name="get-current-action"/></xsl:variable>
    <xsl:variable name="current-lop"><xsl:call-template name="get-current-lop"/></xsl:variable>
    <xsl:variable name="current-max"><xsl:call-template name="get-current-max"/></xsl:variable>
    <xsl:variable name="current-skip"><xsl:call-template name="get-current-skip"/></xsl:variable>
    <xsl:include href="cdml2xsl_utilities.xsl"/>
    <xsl:output method="html" encoding="ISO-8859-1"/>
    <xsl:template match="/fmrs:fmresultset">
        ... the XSLT-CWP statements that were converted from CDML/HTML are inserted here ...
    </xsl:template>
</xsl:stylesheet>
```

- fixes all end-tags to be nested correctly
- changes all element and attribute names to lowercase, but does not change attribute values
- verifies that all attribute values are surrounded by double quotation marks
- assigns a missing attribute value with the attribute name, such as attribute_name="attribute_name"
Conversion of CDML action tags, variable tags, and URLs

To convert CDML action tags, variable tags, and URLs, the CDML Converter does the following:

- The CDML Converter does not change unidentified tag names that are preceded by a dash (–). Unidentified tag names in a URL are moved to the end of the query string. The CDML Converter assumes that tag names that are not preceded by a dash are <fieldname> variable tags and leaves them in the appropriate position in the URL.

- The CDML Converter searches for the text string “fmpro?” to identify URLs. In non-SGML documents and SGML comments, the CDML Converter identifies URLs in any location. Inside SGML elements, the CDML Converter only identifies URLs that are located inside element attribute values.

- The CDML Converter identifies the –format variable tag outside of URLs only if it is inside input elements in HTML documents. The CDML Converter ignores all other form elements, such as text select and text area elements.

- The CDML Converter replaces all <input> elements that contain the CDML –format tag with XSLT-CWP statements. For example, the following <input> element and –format tag:

  `<input type="hidden" name="-format" value="results.htm">`

are replaced with:

  `<xsl:attribute name="action">results.xsl</xsl:attribute>`

- The CDML Converter converts URLs in requests and <form> <input> elements to the new syntax for XSLT. See “About the URL syntax for FileMaker XSLT stylesheets” on page 48.

- The CDML Converter converts URLs that request FileMaker XML data, where –format uses a value of –fmp_xml or –dso_xml to the new URL syntax for XML data. See “About the URL syntax for XML data and container objects” on page 23.

- The CDML Converter converts URLs that request FileMaker image data (using –img) to the URL syntax for requesting container objects in XSLT solutions. See “About the URL syntax for FileMaker container objects in XSLT solutions” on page 48. If a query request contains the img-key format, the CDML Converter converts it in the same way it converts the [FMP-Image] variable tag. For example, this CDML request:

  `<img src="fmpro?-db=employees&key=[FMP-Field: fieldname, url]&-img=">`

is converted the same as this:

  `<img src="[FMP-Image: fieldname]">`

For information on the conversion of [FMP-Image], see “CDML tag name: Image” on page 132.

- The CDML Converter converts tag names to XSLT-CWP query command and parameter names in lowercase, with the exception of <fieldname> CDML variable tags in find requests. Values for query commands and parameters are not changed.

- The CDML Converter reorders the converted XSLT-CWP query commands and parameters in a URL into the following order: –db, –grammar, –lay, any other the query parameters, and the query command last, such as –findall. The CDML Converter does not change the order of <form> <input> elements that contain query commands and parameters.
If a CDML request does not specify a –max query parameter for a –find or –findall query command, the CDML Converter adds –max=25 to the converted XSLT request. The CDML Converter uses the value 25 to create the same behavior as the original CDML request in which 25 records are returned if the –max query parameter is not specified. Note that this behavior has changed in XML or XSLT requests; if the –max query parameter is not specified in an XML or XSLT request, all records are returned. See “–max (Maximum records) query parameter” on page 85.

If a conversion error occurs, the CDML Converter inserts an error comment into the converted stylesheet immediately after the element where the error occurred. You are required to manually fix errors in the stylesheet. An error comment has the following format:

<!-- CDML Converter ERROR: <description of error> -->

If a conversion warning occurs, the CDML Converter inserts a warning comment into the converted stylesheet immediately after the element where the cause of the warning occurred. You are not required to fix warnings in the stylesheet, but you should investigate them to prevent other errors. A warning comment has the following format:

<!-- CDML Converter WARNING: <description of warning> -->

The CDML Converter removes CDML action and variable tags that are obsolete from URLs, and inserts an error comment in the stylesheet where they are removed. See “Conversion of obsolete CDML action tags,” and “Conversion of obsolete CDML variable tags.”

The CDML Converter does not convert the obsolete –mail variable tags (such as –mailto and –mailfrom). See “Conversion of obsolete CDML variable tags” on page 104. To replace the email functionality provided by the –mail variable tags, use an fmxslt:send_email () extension function. See “Sending email messages from the Web Publishing Engine” on page 61.

If a <form> element with an action parameter ending in fmpro does not contain a –format <input> element, the CDML Converter inserts the following error comment:

<!-- CDML Converter ERROR: parameter ‘-format’ not found -->

Conversion of the –error and –errornum CDML variable tags

The CDML Converter converts the CDML –error and –errornum tags into token values. For example, in this original URL:

fmpro?-db=employees&-format=format.htm&-error=error.htm&-errnum=401&-view

the –error tag is converted to –token.error, and the –errnum tag is converted to –token.errnum. The converted XSLT-CWP URL is:

format.xsl?-db=employees&-token.error=error.xsl&-token.errnum=401&-view
The converted format.xsl stylesheet contains the following statements:

```xml
<xsl:include href="cdml2xsl_includes.xsl"/>
<xsl:variable name="_errorcode" select="/fmrs:fmresultset/fmrs:error/@code"/>
<xsl:variable name="token.error" select="$request-query/fmq:query/fmq:parameter[@name = '-token.error']"/>
<xsl:variable name="token.errnum" select="$request-query/fmq:query/fmq:parameter[@name = '-token.errnum']"/>
<xsl:choose>
  <xsl:when test="$token.error != '' and ($_errorcode = $token.errnum)">
    <xsl:call-template name="error"/>
  </xsl:when>
  <xsl:otherwise>
    <!-- ... converted format.xsl code here ... ]
  </xsl:otherwise>
</xsl:choose>
```

The error.html file is converted to an error.xsl stylesheet that contains a named template:

```xml
<xsl:template name="err">
  <!-- ... converted error.xsl code here ... ]
</xsl:template>
```

The CDML Converter includes all error files in a global error file named cdml2xsl_includes.xsl. The following statement includes the global error file in all stylesheets that have error handling:

```xml
<xsl:include href="error.xsl"/>
```

... If an error occurs when the format.xsl stylesheet is processed, and the error code matches the error number specified for the –token.errnum query parameter, the code from error.xsl is executed when format.xsl calls the template named error. This behavior matches the original CDML –error and –errormum functionality.

### Conversion of obsolete CDML action tags

Here’s how the CDML Converter handles the conversion of the following obsolete CDML action tags:

<table>
<thead>
<tr>
<th>This obsolete CDML action tag</th>
<th>Is</th>
</tr>
</thead>
<tbody>
<tr>
<td>-dbclose</td>
<td>Replaced by this comment after the original action tag location: &lt;!-- CDML Converter ERROR: '-dbclose' not supported by XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>-dbopen</td>
<td>Replaced by this comment after the original action tag location: &lt;!-- CDML Converter ERROR: '-dbopen' not supported by XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>-img</td>
<td>Removed and the specified URL for the image is converted to new URL syntax for container objects. See “About the URL syntax for FileMaker container objects in XML solutions,” and “About the URL syntax for FileMaker container objects in XSLT solutions” on page 48.</td>
</tr>
</tbody>
</table>
**Conversion of supported CDML action tags**

Here’s how the CDML Converter handles the conversion of the following supported CDML action tags:

<table>
<thead>
<tr>
<th>This CDML action tag</th>
<th>Is converted to this XSLT-CWP query parameter with no change in syntax</th>
</tr>
</thead>
<tbody>
<tr>
<td>-delete</td>
<td>-delete</td>
</tr>
<tr>
<td>-dup</td>
<td>-dup</td>
</tr>
<tr>
<td>-edit</td>
<td>-edit</td>
</tr>
<tr>
<td>-find</td>
<td>-find</td>
</tr>
<tr>
<td>-findall</td>
<td>-findall</td>
</tr>
<tr>
<td>-findany</td>
<td>-findany</td>
</tr>
<tr>
<td>-new</td>
<td>-new</td>
</tr>
<tr>
<td>-view</td>
<td>-view</td>
</tr>
<tr>
<td>-dbnames</td>
<td>-dbnames</td>
</tr>
<tr>
<td>-layoutnames</td>
<td>-layoutnames</td>
</tr>
<tr>
<td>-scriptnames</td>
<td>-scriptnames</td>
</tr>
</tbody>
</table>

**Conversion of obsolete CDML variable tags**

Here’s how the CDML Converter handles the conversion of the following obsolete CDML variable tags:

<table>
<thead>
<tr>
<th>This obsolete CDML variable tag</th>
<th>Is replaced with this comment after the original variable tag location</th>
</tr>
</thead>
<tbody>
<tr>
<td>-errorfmtfield</td>
<td>&lt;!-- CDML Converter ERROR: '-errorfmtfield' not supported by XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>-fmtfield</td>
<td>&lt;!-- CDML Converter ERROR: '-fmtfield' not supported by XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>-format</td>
<td>A comment is not inserted. Instead, the URL to the format file is converted to the new URL syntax to refer to the converted XSLT stylesheet name</td>
</tr>
<tr>
<td>-mailbcc</td>
<td>&lt;!-- CDML Converter ERROR: '-mailbcc' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
</tr>
<tr>
<td>-mailcc</td>
<td>&lt;!-- CDML Converter ERROR: '-mailcc' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
</tr>
<tr>
<td>-mailfmtfield</td>
<td>&lt;!-- CDML Converter ERROR: '-mailfmtfield' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
</tr>
<tr>
<td>-mailformat</td>
<td>&lt;!-- CDML Converter ERROR: '-mailformat' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
</tr>
<tr>
<td>-mailfrom</td>
<td>&lt;!-- CDML Converter ERROR: '-mailfrom' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
</tr>
<tr>
<td>-mailhost</td>
<td>&lt;!-- CDML Converter ERROR: '-mailhost' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
</tr>
<tr>
<td>-mailsub</td>
<td>&lt;!-- CDML Converter ERROR: '-mailsub' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
</tr>
</tbody>
</table>
## Conversion of supported CDML variable tags

Here’s how the CDML Converter handles the conversion of the following supported CDML variable tags:

<table>
<thead>
<tr>
<th>This CDML variable tag</th>
<th>Is converted to this XSLT-CWP query parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-mailto</td>
<td>&lt;!-- CDML Converter ERROR: '-mailto' not supported by XSLT-CWP, use fmxslt:send_email() function --&gt;</td>
<td></td>
</tr>
<tr>
<td>-password</td>
<td>&lt;!-- CDML Converter ERROR: '-password' not supported by XSLT-CWP --&gt;</td>
<td></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>This CDML variable tag</th>
<th>Is converted to this XSLT-CWP query parameter</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>-db</td>
<td>-db</td>
<td>no change to syntax; all occurrences of .fp5 filename extension are removed</td>
</tr>
<tr>
<td>-lay</td>
<td>-lay</td>
<td>no change to syntax</td>
</tr>
<tr>
<td>-lop</td>
<td>-lop</td>
<td>no change to syntax; the “and” and “or” keywords are changed to lowercase</td>
</tr>
<tr>
<td>-max</td>
<td>-max</td>
<td>no change to syntax; if -max was not specified in CDML, the CDML Converter inserts -max=25 into the converted XSLT</td>
</tr>
<tr>
<td>-modid</td>
<td>-modid</td>
<td>no change to syntax</td>
</tr>
<tr>
<td>-op</td>
<td>field.op</td>
<td>Changed to support the new syntax for find operators (field.op). See “fieldname.op (Comparison operator) query parameter” on page 83.</td>
</tr>
<tr>
<td>-recid</td>
<td>-recid</td>
<td>no change to syntax</td>
</tr>
<tr>
<td>-script</td>
<td>-script</td>
<td>This comment is added after element: &lt;!-- CDML2XSLT WARNING: please verify '-script' functionality with XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>-script.prefind</td>
<td>-script.prefind</td>
<td>no change to syntax. Check '-script.prefind' functionality in XSLT-CWP.</td>
</tr>
<tr>
<td>-script.presort</td>
<td>-script.presort</td>
<td>no change to syntax. Check '-script.presort' functionality in XSLT-CWP.</td>
</tr>
<tr>
<td>-skip</td>
<td>-skip</td>
<td>no change to syntax</td>
</tr>
<tr>
<td>-sortfield</td>
<td>-sortfield</td>
<td>Changed to support the new syntax for sort operators (~sortfield.precedence). See “~sortfield (Sort field) query parameter” on page 86.</td>
</tr>
<tr>
<td>-sortorder</td>
<td>-sortorder</td>
<td>Changed to support the new syntax for sort operators (~sortorder.precedence). See “~sortorder (Sort order) query parameter” on page 87.</td>
</tr>
<tr>
<td>-styletype</td>
<td>-styletype</td>
<td>no change to syntax</td>
</tr>
<tr>
<td>-stylehref</td>
<td>-stylehref</td>
<td>no change to syntax</td>
</tr>
</tbody>
</table>
Conversion of CDML boolean parameters to XPath boolean parameters

CDML boolean expressions, which are used within [FMP-If] and [FMP-ElseIf] intratag parameters, are composed of parameters and operators that the CDML Converter converts to XPath expressions.

<table>
<thead>
<tr>
<th>This CDML boolean parameter</th>
<th>Is converted to this XPath boolean parameter</th>
<th>Comments</th>
</tr>
</thead>
<tbody>
<tr>
<td>CanDelete</td>
<td>false()</td>
<td>For security reasons, this parameter is converted to false(), but it is not supported. This comment is added after tag: &lt;!-- CDML Converter ERROR: 'CanDelete' not supported by XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>CanEdit</td>
<td>false()</td>
<td>For security reasons, this parameter is converted to false(), but it is not supported. This comment is added after tag: &lt;!-- CDML Converter ERROR: 'CanEdit' not supported by XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>CanNew</td>
<td>false()</td>
<td>For security reasons, this parameter is converted to false(), but it is not supported. This comment is added after tag: &lt;!-- CDML Converter ERROR: 'CanNew' not supported by XSLT-CWP --&gt;</td>
</tr>
<tr>
<td>IsSorted</td>
<td>boolean($request-query/fmq:query/ fmq:parameter[starts-with(@name, '-sortfield')] = true())</td>
<td></td>
</tr>
<tr>
<td>True</td>
<td>true()</td>
<td></td>
</tr>
<tr>
<td>False</td>
<td>false()</td>
<td></td>
</tr>
<tr>
<td>Checked</td>
<td>false()</td>
<td></td>
</tr>
<tr>
<td>String</td>
<td>'String'</td>
<td></td>
</tr>
<tr>
<td>Number</td>
<td>Number</td>
<td></td>
</tr>
<tr>
<td>&lt;missing parameter&gt;</td>
<td>&quot;</td>
<td>[FMP-If xyz .aq. ]/[FMP-If] is changed to: <a href="">xsl:choose</a>&lt;xsl:when test=&quot;xyz = '&quot;&quot;&quot;&gt; ... &lt;/xsl:when&gt; &lt;/xsl:choose&gt;</td>
</tr>
</tbody>
</table>

Note If the intratag parameters CurrentDate, CurrentDay, or CurrentTime are used in a boolean expression, the CDML Converter replaces them with the appropriate XSLT-CWP date comparison extension function, such as fmxslt:compare_date(). See “Using the date, time, and day extension functions” on page 66.
Converting CDML solutions to FileMaker XSL

Conversion of CDML boolean operators to XPath

<table>
<thead>
<tr>
<th>This CDML boolean operator</th>
<th>Is converted to this XPath boolean operator</th>
</tr>
</thead>
<tbody>
<tr>
<td>x .and. y</td>
<td>x and y</td>
</tr>
<tr>
<td>x .or. y</td>
<td>x or y</td>
</tr>
<tr>
<td>x .xor. y</td>
<td>(x or y) and not (x and y)</td>
</tr>
<tr>
<td>x .eq. y</td>
<td>x = y</td>
</tr>
<tr>
<td>x .neq. y</td>
<td>x != y</td>
</tr>
<tr>
<td>x .gt. y</td>
<td>x &gt; y</td>
</tr>
<tr>
<td>x .gte. y</td>
<td>x &gt;= y</td>
</tr>
<tr>
<td>x .lt. y</td>
<td>x &lt; y</td>
</tr>
<tr>
<td>x .lte. y</td>
<td>x &lt;= y</td>
</tr>
<tr>
<td>x .cn. y</td>
<td>contains(x, y)</td>
</tr>
<tr>
<td>x .ncn. y</td>
<td>not (contains(x, y))</td>
</tr>
</tbody>
</table>

Conversion of CDML intratag parameters to XSLT-CWP

CDML intratag parameters are CDML replacement tags that are used within a parameter of another replacement tag. If an intratag parameter is used as a parameter value, it is surrounded by curly braces. For example:

```
-db={CurrentDatabase}
```

instead of:

```
[FMP-CurrentDatabase]
```

The CDML Converter converts intratag parameters in the same way it converts replacement tags. See “Conversion of CDML replacement tags to XSLT-CWP” on page 111. Here is an example of how the CDML Converter handles the conversion of intratag parameters used as parameter values.

Example of intratag parameter conversion inside parameter values (shown in curly braces):

```
Original CDML: <!-- Log page hits in another database -->
[FMP-InlineAction: -db=log.fp5, -lay=web, time="{CurrentTime}",
date="{CurrentDate}",
page="This page!", browser="{ClientType}", ip="{ClientIP}", -new]
[/FMP-InlineRequest]

Converted XSLT-CWP: <!-- Log page hits in another database -->
<xsl:variable select="document(concat('/fmi/xml/fmresultset.xml?-db=', fmxslt:url_encode('log'), '&-lay=',
fmxslt:url_encode('web'), '&time=', fmxslt:url_encode(fmxslt:get_time()), '&date=',
fmxslt:url_encode(fmxslt:get_date()), '&page=', fmxslt:url_encode('This Page!'), '&browser=',
fmxslt:url_encode(fmxslt:get_header('User-Agent')), '&ip=', fmxslt:url_encode($client-ip), '&-new=')) />
```
**Manually fixing CDML conversion errors**

There are some situations where the CDML Converter cannot automatically determine the correct conversion from CDML to XSLT. For these types of situations, you must use a text editor or XSLT stylesheet editor to manually fix the problems in the converted XSLT stylesheets:

- **In FileMaker Pro**, script names are limited to 100 characters. During database conversion, any script names over this limit are truncated in the database. Since the CDML Converter does not truncate script names over this limit, you must change the script name in the XSLT stylesheet to match the truncated name in the database.

- The CDML Converter uses the configured locale on the host computer where the Web Publishing Engine is installed when inserting server-side functions in the XSLT stylesheets, such as `fmxslt:get_date()` and `fmxslt:get_time()`. If your CDML format files include date and time formatting strings that are passed to these functions, you must manually localize the strings after the conversion, such as changing Month/Day/Year to Day/Month/Year.

- If a CDML solution contains a form that provides a user-selectable value for the –format tag, such as a choice of format files, the CDML Converter cannot automatically create a functionally equivalent XSLT solution. You must manually edit the converted XSLT solution with a separate request for each XSLT stylesheet you want to offer web users.

- If your CDML solution contains query requests that use the –lay tag to specify a layout, make sure the specified layouts contain all of the fields referenced in the requests. If the specified layout does not contain all of the fields, you must either add the fields to the layout or manually change the –lay query parameter to refer to a layout that does contain all of the fields. Alternatively, if you need to submit a request using a layout which contains fields that don’t exist in another layout, you can use the –lay.response query parameter to switch layouts. See “Switching layouts for an XML response” on page 35.

- If your CDML solution contains query requests that do not use the –lay tag to specify a layout, the CDML Converter automatically adds a –lay query parameter to the requests and specifies a value of AllFieldsLayout. You must either manually change the value for the –lay parameter in the converted stylesheets to match the layout you want to use in your database, or add a layout called AllFieldsLayout to your database.

- If your CDML solution contains -script, -script.prefind, or -script.presort variable tags, check the script functionality in the converted XSLT stylesheet.

- In CDML, field and database name comparisons were case-insensitive, which allowed you to use a tag such as `[FMP-Field:myfield]` to refer to a field named MyField or myField. In XSLT-CWP, field and database name comparisons are case-sensitive if they are not used in a query string. In the converted stylesheets, you must manually fix any field and database names in XSLT statements (excluding query strings) to exactly match the names used in the database solution, including the case of the name.

For example, in this statement:

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

the field reference `LastName` must exactly match the name and case of the LastName field in the database.

**Note** In XSLT-CWP, field and database names used in query strings are case-insensitive. See “Guidelines for using query commands and parameters” on page 76.
In CDML, you could compare fields without including the field attribute. For example, you could use either [FMP-If: myfield.eq.10] or [FMP-If: field:myfield.eq.10]. In this example, because the field attribute is not included in the comparison, the CDML Converter converts myfield as a string literal instead of a field name. For example, after conversion, this CDML statement:

[FMP-If: myfield.eq.10]

is converted to these XSLT-CWP statements:

```xml
<xsl:choose>
  <xsl:when test="myfield" = '10'>Ten</xsl:when>
</xsl:choose>
```

To fix this type of problem, you must manually fix the statement to have the appropriate field name in the comparison statements in the converted stylesheet. Alternatively, you can add "field:" to the CDML file where required and reconvert the file.

The CDML Converter fixes several instances of malformed HTML that are generated by Claris Home Page, assuming that the metatags are included in the beginning of the HTML page. If the metatags have been removed, the CDML Converter will not fix the malformed HTML. There may be other instances of malformed HTML in Claris Home Page or other HTML files that the CDML Converter cannot fix and convert properly to XHTML, which is more strictly constructed than HTML. In cases where malformed HTML was not converted correctly, you must manually fix the XHTML in the converted stylesheet.

The CDML Converter adds a .jpg filename extension to all image filename references when it converts --img action tags. For example, the CDML Converter converts this request:

```
/fmpro?-db=products&-format=format_file.html&-lay=sales&-recid=123&-img
```

to this XSLT-CWP request:

```
/fmri/xsl/data.jpg?-db=products&-lay=sales&-recid=123
```

If the .jpg filename extension is incorrect for your solution, you must manually change the extension in filename references in the converted stylesheets.

The CDML Converter cannot convert nested (embedded) forms. If you nest forms, you need to either change your CDML solution or fix the resulting .xsl file.

The CDML Converter cannot convert CDML tags that are embedded in an SGML tag, such as in this example:

```xml
<input type="text" NAME="StateProvince" SIZE="16" [FMP-If:currentdatabase.eq.Customers.fp5]
VALUE="[FMP-Field:StateProvince]" [/FMP-If]>
         <SELECT NAME="-max">
            <OPTION [FMP-If:currentmax.eq.5] SELECTED[/FMP-If]>5
            <OPTION [FMP-If:currentmax.eq.10] SELECTED[/FMP-If]>10
        </SELECT>
```
To avoid conversion problems resulting from the preceding example, change the CDML source code as follows:

[FMP-If: currentdatabase.eq.Customers.fp5]
   <input type="text" NAME="StateProvince" SIZE="16" VALUE="[FMP-Field:StateProvince]">
[FMP-Else]
   <input type="text" NAME="StateProvince" SIZE="16" VALUE="">
[/FMP-If]
   <SELECT NAME="-max">
       [FMP-If:currentmax.eq.5]
           <OPTION SELECTED>5 </OPTION>
       [FMP-Else]
           <OPTION>5 </OPTION>
       [FMP-If:currentmax.eq.10]
           <OPTION SELECTED>10  </OPTION>
       [FMP-Else]
           <OPTION>10 </OPTION>
   [/FMP-If]
</SELECT>

The CDML Converter removes instances of the –format tag in a form and converts its value into an attribute of the form tag. The result in the converted stylesheet is that there is one less input parameter in the form. This may cause problems if your CDML solution uses JavaScript to refer to the form fields.

For example, the following form:

<FORM METHOD="POST" ACTION="FMPro" NAME="checkoutform">
   <INPUT TYPE="hidden" NAME="-db" VALUE="Orders.FP5">
   <INPUT TYPE="hidden" NAME="-format" VALUE="thanks.htm">
   <INPUT TYPE="hidden" NAME="-lay" VALUE="CGI">
   <INPUT TYPE="text" SIZE="50" NAME="Account Number">
</FORM>

is converted to these XSLT statements:

<form method="POST" name="checkoutform" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">
   <input type="hidden" name="-db" value="Orders"/>
   <input type="hidden" name="-format" value="thanks.htm"/>
   <input type="hidden" name="-lay" value="CGI"/>
   <input type="text" size="50" name="Account Number"/>
</form>

The problem is that JavaScript is used to make sure that the user submits a value in the Account Number field:

<P>
   <CENTER>
       <A HREF="javascript:document.checkoutform.elements[3].value == '' ? alert('You must fill out the \Account number\ field!') : document.checkoutform.submit()">
           <IMG SRC="images/continue1.gif" NAME="cont" ALT="Continue"></A>
   </CENTER>
</P>
After conversion of this example, there are fewer elements in the checkoutform because the –format tag has been removed. To fix this example, you must manually change the JavaScript to check document.checkoutform.elements[2] rather than document.checkoutform.elements[3].

- The expression evaluation logic in CDML is different from the logic in XSLT. For example, the following CDML expression outputs “userchoice less than 1” if userchoice is empty:

  ```cdml
  [fmp-if: currentcookie:userchoice .lt. 1 ]
  userchoice less than 1
  [fmp-else]
  userchoice not less than 1
  [/fmp-if]
  ```

In XSLT, the same converted expression outputs “userchoice not less than 1” if userchoice is empty:

```xml
<xsl:choose>
  <xsl:when test="fmxslt:get_cookie('userchoice') &lt; '1'">
    userchoice less than 1
  </xsl:when>
  <xsl:otherwise>
    userchoice not less than 1
  </xsl:otherwise>
</xsl:choose>
```

The reason for the difference is that the CDML expression performs a numeric comparison, but the XSLT expression performs a string comparison. To return the same results in XSLT as the original CDML expression, change the XSLT statements to account for the empty string. For example:

```xml
<xsl:variable name="userchoice" select="fmxslt:get_cookie('userchoice')"/>
<xsl:choose>
  <xsl:when test="string-length($userchoice) = 0 or $userchoice &lt; '1'">
    userchoice less than 1
  </xsl:when>
  <xsl:otherwise>
    userchoice not less than 1
  </xsl:otherwise>
</xsl:choose>
```

### Conversion of CDML replacement tags to XSLT-CWP

This section describes the conversion of all CDML replacement tags to XSLT-CWP statements. In all cases:

- The converted XSLT-CWP statements use the fmresultset grammar.
- When an XSLT-CWP statement returns a value inside an SGML element attribute value, the {name} form is used; otherwise, the `<xsl:value-of select="name" />` form is used.
- When a CDML tag has an Encoding parameter, the converted XSLT-CWP value is passed through one of the following extension functions for encoding: `fmxslt:break_encode()`, `fmxslt:html_encode()`, or `fmxslt:url_encode()`. See “Using the string manipulation extension functions” on page 64. The Encoding value “Raw” specifies no encoding. When the encoding is set to “Raw”, the CDML Converter sets the “disable-output-escaping” attribute of the `<xsl:value-of>` and `<xsl:text>` elements to “yes”.
- When inside an HTML-type document, the `fmxslt:html_encode()` extension function is not used.
CDML tag name: Client Address
Tag is replaced with the current web client’s IP address
CDML syntax: [FMP-ClientAddress]
XSLT-CWP conversion:
- when inside SGML element attribute value: ${client-ip}
- otherwise: <xsl:value-of select="$client-ip"/>

Conversion example
<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Your address is: [FMP-ClientAddress]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted XSLT-CWP:</td>
<td>Your address is: &lt;xsl:value-of select=&quot;$client-ip&quot;/&gt;</td>
</tr>
<tr>
<td>Converted Result:</td>
<td>Your address is: 192.168.123.101</td>
</tr>
</tbody>
</table>

CDML tag name: Client IP Address
Tag is replaced with current client’s IP address
CDML syntax: [FMP-ClientIP]
XSLT-CWP conversion:
- when inside SGML element attribute value: ${client-ip}
- otherwise: <xsl:value-of select="$client-ip"/>

Conversion example
<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Your IP address is: [FMP-ClientIP]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted XSLT-CWP:</td>
<td>Your IP address is: &lt;xsl:value-of select=&quot;$client-ip&quot;/&gt;</td>
</tr>
<tr>
<td>Converted Result:</td>
<td>Your IP address is: 192.168.123.101</td>
</tr>
</tbody>
</table>

CDML tag name: Client Password
Tag is replaced with the current HTTP-authenticated client password
CDML syntax: [FMP-ClientPassword]
XSLT-CWP conversion:
- when inside SGML element attribute value: ${client-password}
- otherwise: <xsl:value-of select="$client-password"/>

Conversion example
<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Your password is: [FMP-ClientPassword]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted XSLT-CWP:</td>
<td>Your password is: &lt;xsl:value-of select=&quot;$client-password&quot;/&gt;</td>
</tr>
<tr>
<td>Converted Result:</td>
<td>Your password is: my-password</td>
</tr>
</tbody>
</table>
**CDML tag name: Client Type**
Tag is replaced with the current browser client type

**CDML syntax:** [FMP-ClientType]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{fmxslt:get_header('User-Agent')}`
- otherwise: `<xsl:value-of select="fmxslt:get_header('User-Agent')"/>

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
</table>
| Your browser type is: [FMP-ClientType] | Your browser type is: `<xsl:value-of select="fmxslt:get_header('User-Agent')"/>
| | Mozilla/3.01 (Macintosh; I; PPC) |

**CDML tag name: Client User Name**
Tag is replaced with the client’s user name from HTTP authentication

**CDML syntax:** [FMP-ClientUserName]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{$client-user-name}`
- otherwise: `<xsl:value-of select="$client-user-name"/>

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
</table>
| Your name is: [FMP-ClientUserName] | Your name is: `<xsl:value-of select="$client-user-name"/>
| | my-user-name |

**CDML tag name: Content MIME Type**
Tag is replaced with nothing in the HTML, and changes the MIME type returned to the browser

**CDML syntax:** [FMP-ContentMIMEType: MimeType]

**XSLT-CWP conversion:**
- `<xsl:variable name="header1" select="fmxslt:set_header('Content-Type', 'MimeType')"/>
- when inside SGML element attribute value: insert outside/after the element

**Note** Because the `fmxslt:set_header()` extension function is used after conversion, content buffering must be enabled. To enable content buffering, the CDML Converter automatically inserts the `<?xslt-cwp-buffer?>` processing instruction. See “Using content buffering” on page 58.

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
</table>
| Some sample [FMP-ContentMIMEType: text/plain] text. | Some sample `<xsl:variable name="header1" select="fmxslt:set_header('Content-Type', 'text/plain')"/> text. | Some sample text.
| | (The HTTP response also contains **Content-Type: text/plain** in the header.) |
**CDML tag name: Cookie**

Tag is replaced with the current value of the specified cookie

**CDML syntax:** [FMP-Cookie: CookieName, Encoding], where Encoding is 'Raw' (default), or 'URL'

**XSLT-CWP conversion:**
- when inside SGML element attribute value: {fmxslt:get_cookie('CookieName')}
- otherwise: <xsl:value-of select="fmxslt:get_cookie('CookieName')"/>

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Thanks for coming back! The latest products in [FMP-Cookie: ColorChoice] are listed below</td>
<td>Thanks for coming back! The latest products in &lt;xsl:value-of select=&quot;fmxslt:get_cookie('ColorChoice')&quot;/&gt; are listed below.</td>
<td>Thanks for coming back! The latest products in green are listed below.</td>
</tr>
</tbody>
</table>

**CDML tag name: Current Action**

Tag is replaced with the name of the current action, such as: find, findall, new, edit, delete, view, or dup. The returned name does not include the dash (–) character.

**CDML syntax:** [FMP-CurrentAction: Encoding], where Encoding is 'HTML' (default), or 'Display'

**XSLT-CWP conversion:**
- when inside SGML element attribute value: {$current-action}
- otherwise: <xsl:value-of select="$current-action"/>
- if Format is 'Display': <!-- CDML2XSLT WARNING: [FMP-CurrentAction] 'Display' Encoding not supported by XSLT-CWP -->
- if used in document, $current-action-variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Last action was: [FMP-CurrentAction]</td>
<td>Last action was: &lt;xsl:value-of select=&quot;$current-action&quot;/&gt;</td>
<td>Last action was: view</td>
</tr>
</tbody>
</table>
**CDML tag name: Current Database**

Tag is replaced with the name of the database being processed

**CDML syntax:** \[FMP-CurrentDatabase: Encoding\], where *Encoding* is 'Raw', 'URL', or 'HTML' (default).

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{fmrs:fmresultset/fmrs:datasource/@database}`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>You are viewing the [FMP-CurrentDatabase] database.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted Result:</td>
<td>You are viewing the <strong>Contact</strong> database.</td>
</tr>
</tbody>
</table>

**CDML tag name: Current Date**

Tag is replaced with the current date

**CDML syntax:** \[FMP-CurrentDate: Format\], where *Format* is 'Short' (default) or 'Long'.

**XSLT-CWP conversion:**
- when inside SGML element attribute value, 'Short': `{fmxslt:get_date('short')}`
- or, 'Long': `{fmxslt:get_date('long')}`
- otherwise, 'Short': `<xsl:value-of select="fmxslt:get_date('short')"/>`
- or, 'Long': `<xsl:value-of select="fmxslt:get_date('long')"/>`
- if *Format* is 'Abbrev': <!-- CDML CONVERTER WARNING: Abbreviated date format is no longer supported. Use fmxslt:get_datetime() for additional formatting-->`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Current date is: [FMP-CurrentDate]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted XSLT-CWP:</td>
<td>Current date is: <code>&lt;xsl:value-of select=&quot;fmxslt:get_date()&quot;/&gt;</code></td>
</tr>
<tr>
<td>Converted Result:</td>
<td>Current date is: <strong>2/2/04</strong></td>
</tr>
</tbody>
</table>
**CDML tag name: Current Day**

Tag is replaced with the name of the current day of the week

**CDML syntax**: `[FMP-CurrentDay: Format]`, where `Format` is 'Short' (default), or 'Long'

**XSLT-CWP conversion**:

- when inside SGML element attribute value, 'Short': `{fmxslt:get_day('short')}
- or, 'Long': `{fmxslt:get_day('long')}
- otherwise, 'Short': `<xsl:value-of select="fmxslt:get_day('short')" />`
- or, 'Long': `<xsl:value-of select="fmxslt:get_day('long')" />`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current is: [FMP-CurrentDay: Long]</td>
<td>Current day is: <code>&lt;xsl:value-of select=&quot;fmxslt:get_day('long')&quot; /&gt;</code></td>
<td>Current day is: Monday</td>
</tr>
</tbody>
</table>

---

**CDML tag name: Current Error**

Tag is replaced with the FileMaker error number from the current action

**CDML syntax**: `[FMP-CurrentError]`

**XSLT-CWP conversion**:

when inside SGML element attribute value: `{fmrs:fmresultset:/fmrs:error/@code}`

otherwise: `<xsl:value-of select="fmrs:fmresultset:/fmrs:error/@code" />`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The last operation failed with error number [FMP-CurrentError].</td>
<td>The last operation failed with error number <code>&lt;xsl:value-of select=&quot;fmrs:fmresultset/fmrs:error/@code&quot; /&gt;</code>.</td>
<td>The last operation failed with error number 500.</td>
</tr>
</tbody>
</table>

---

**CDML tag name: Current Find**

Repeats the HTML between `[FMP-CurrentFind]` and `[FMP-CurrentFind]` for each find criteria that was part of the request that created this page

**CDML syntax**: `[FMP-CurrentFind]...[/FMP-CurrentFind]`
XSLT-CWP conversion:

- `<xsl:for-each select="$current-find">...`<xsl:for-each>
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-CurrentFind] not in a valid location -->
- if used in document, $current-find variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet

Conversion example

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
</table>
| Current find request is:<br><br>  
[FMP-CurrentFind]  
Field: [FMP-FindFieldItem], Op: [FMP-FindOpItem], Value: [FMP-FindValueItem]<br><br>[FMP-CurrentFind]  
Field: [FMP-FindFieldItem], Op: [FMP-FindOpItem], Value: [FMP-FindValueItem]|<xsl:for-each select="$current-find/find-field">  
Field: First Name, Op: begins with, Value: John<br><br>Field: Last Name, Op: equals, Value: Doe<br><br>

**CDML tag name: Current Format File**

Tag is replaced with the name of the current format file being processed

**CDML syntax:** [FMP-CurrentFormat: Encoding], where Encoding is 'Raw', 'URL', or 'HTML' (default)

**XSLT-CWP conversion:**

- when inside SGML element attribute value: `{request-query/@action}`
- otherwise: `<xsl:value-of select="$request-query/@action"/>`

Conversion example

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>This page was formatted with file: [FMP-CurrentFormat].</td>
<td>This page was formatted with file: &lt;xsl:value-of select=&quot;$request-query/@action&quot;/&gt;.</td>
<td>This page was formatted with file: Detail.xsl.</td>
</tr>
</tbody>
</table>
**CDML tag name: Current Found Count**
Tag is replaced with the total number of records in the current found set

**CDML syntax:** [FMP-CurrentFoundCount]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{fmrs:fmresultset/fmrs:resultset/@count}`
- otherwise: `<xsl:value-of select="fmrs:fmresultset/fmrs:resultset/@count" />`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Record(s) in found set: [FMP-CurrentFoundCount]</td>
<td>Record(s) in found set: <code>&lt;xsl:value-of select=&quot;fmrs:fmresultset/fmrs:resultset/@count&quot; /&gt;</code></td>
<td>Record(s) in found set: 12</td>
</tr>
</tbody>
</table>

**CDML tag name: Current Layout**
Tag is replaced with the name of the layout used to process the page

**CDML syntax:** [FMP-CurrentLayout: Encoding], where Encoding is 'Raw', 'URL', or 'HTML' (default)

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{fmrs:fmresultset/fmrs:datasource/@layout}`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>This page was formatted using: [FMP-CurrentLayout] layout.</td>
<td>This page was formatted using: <code>&lt;xsl:value-of select=&quot;fmrs:fmresultset/fmrs:datasource/@layout&quot; /&gt;</code> layout.</td>
<td>This page was formatted using: Detail layout.</td>
</tr>
</tbody>
</table>

**CDML tag name: Current Logical Operator**
Tag is replaced with the logical operator used for the current search

**CDML syntax:** [FMP-CurrentLOP]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{current-lop}`
- otherwise: `<xsl:value-of select="$current-lop" />`
- if used in document, $current-lop variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>You just did an &quot;[FMP-CurrentLOP]&quot; search.</td>
<td>You just did an &quot;&lt;xsl:value-of select=&quot;$current-lop&quot; /&gt;&quot; search.</td>
<td>You just did an &quot;or&quot; search.</td>
</tr>
</tbody>
</table>
**CDML tag name: Current Max**

Tag is replaced with the maximum number of records that was specified

**CDML syntax:** [FMP-CurrentMax]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{$current-max}
- otherwise: `<xsl:value-of select="$current-max" />`
- if used in document, `$current-max` variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet.

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
<tbody>
<tr>
<td>Click the button to see the next [FMP-CurrentMax] records.</td>
<td>Click the button to see the next <code>&lt;xsl:value-of select=&quot;$current-max&quot; /&gt;</code> records.</td>
<td>Click the button to see the next 10 records.</td>
</tr>
</tbody>
</table>

**CDML tag name: Current Modification ID**

Tag is replaced with the modification ID of the current record being edited

**CDML syntax:** [FMP-CurrentModID]

**XSLT-CWP conversion:**
- when inside SGML element attribute value:
  - when current context is a record: `{@mod-id}
  - otherwise: `/fmrs:fmresultset/fmrs:resultset/fmrs:record[1]/@mod-id`
- otherwise:
  - when current context is a record: `<xsl:value-of select="@mod-id" />

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>&lt;FORM ACTION=&quot;FMPro&quot; METHOD=&quot;POST&quot;&gt; &lt;INPUT TYPE=&quot;HIDDEN&quot; NAME=&quot;-DB&quot; VALUE=&quot;contacts.fp5&quot;&gt; &lt;INPUT TYPE=&quot;HIDDEN&quot; NAME=&quot;-Format&quot; VALUE=&quot;results.htm&quot;&gt; &lt;INPUT TYPE=&quot;HIDDEN&quot; NAME=&quot;-RecID&quot; VALUE=&quot;[FMP-CurrentRecID]&quot;&gt; &lt;INPUT TYPE=&quot;HIDDEN&quot; NAME=&quot;-ModID&quot; VALUE=&quot;[FMP-CurrentModID]&quot;&gt; &lt;INPUT TYPE=&quot;TEXT&quot; NAME=&quot;Country&quot;&gt; &lt;INPUT TYPE=&quot;SUBMIT&quot; NAME=&quot;-Edit&quot; VALUE=&quot;Edit This Record&quot;&gt; &lt;/FORM&gt;</td>
<td></td>
</tr>
</tbody>
</table>
Conversion example

**CDML tag name: Current Portal Number**

Tag is replaced with the row number of the current portal being processed. This is always found within a [FMP-Portal] loop.

**CDML syntax:** [FMP-CurrentPortalRowNumber]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{position()}`
- otherwise: `<xsl:value-of select="position()"/>

**Conversion example**

**Original CDML:**

```xml
[FMP-Portal: lineitems]
[FMP-CurrentPortalRowNumber]: [FMP-Field: lineitems::name]<br>
[/FMP-Portal]
```

**Converted XSLT-CWP:**

```xml
<xsl:for-each select="fmrs:relatedset[@table='lineitems']/fmrs:record">
  <xsl:value-of select="position()"/>
  <xsl:value-of select="fmrs:field[@name = 'name']"/>
  <br />
</xsl:for-each>
```

**Converted Result:**

```
1: Red<br>
```
**CDML tag name: Current Record Count**

Tag is replaced with the total number of records in the database

**CDML syntax:** [FMP-CurrentRecordCount]

**XSLT-CWP conversion:**

- when inside SGML element attribute value: `{fmrs:fmresultset/fmrs:datasource/@total-count}`
- otherwise: `<xsl:value-of select="fmrs:fmresultset/fmrs:datasource/@total-count"/>

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Total number of records in the database is: [FMP-CurrentRecordCount]</td>
<td>Total number of records in the database is: <code>&lt;xsl:value-of select=&quot;fmrs:fmresultset/fmrs:datasource/@total-count&quot;/&gt;</code></td>
</tr>
<tr>
<td>Converted Result:</td>
<td>Total number of records in the database is: 1123</td>
</tr>
</tbody>
</table>

**CDML tag name: Current Record ID**

Tag is replaced with the record ID

**CDML syntax:** [FMP-CurrentRecID]

**XSLT-CWP conversion:**

- when inside SGML element attribute value:
  - when current context is a record: `{@record-id}`
  - otherwise: `/{fmrs:fmresultset/fmrs:resultset/fmrs:record[1]/@record-id}`
- otherwise:
  - when current context is a record: `<xsl:value-of select="@record-id" />`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
</tr>
</thead>
</table>
| <form action="FMPro" method="post">  
  <input type="hidden" name="-DB" value="name.fp5">  
  <input type="hidden" name="-Format" value="results.htm">  
  <input type="hidden" name="-RecID" value="[FMP-CurrentRecID]">  
  <input type="submit" name="-Delete" value="Delete This Record">  
</form> | <form action="/fmi/xsl/results.xsl" method="post">  
  <input type="hidden" name="-DB" value="name"></input>  
  <input type="hidden" name="-grammar" value="fmresultset"></input>  
  <input type="hidden" name="-RecID" value="[@record-id]"></input>  
  <input type="submit" name="-Delete" value="Delete This Record"></input>  
</form> |
**Conversion example**

**Converted Result:**

```html
<form action="/fmi/xsl/results.xsl" method="post">
  <input type="hidden" name="-DB" value="name">
  <input type="hidden" name="-grammar" value="fmresultset">
  <input type="hidden" name="-RecID" value="1023">
  <input type="submit" name="-Delete" value="Delete This Record">
</form>
```

**CDML tag name: Current Record Number**

Tag is replaced with the record’s position in the current found set

**CDML syntax:** [FMP-CurrentRecordNumber]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{position()}`
- otherwise: `<xsl:value-of select="position() + $current-skip" />`
- when outside of a [FMP-Record] loop: `<!-- CDML Converter ERROR: [FMP-CurrentRecordNumber] outside of [FMP-Record] -->`
- if used in document, $current-skip variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current record in found set: [FMP-CurrentRecordNumber]</td>
<td>Current record in found set: <code>&lt;xsl:value-of select=&quot;position() + $current-skip&quot; /&gt;</code></td>
</tr>
<tr>
<td>Converted Result: Current record in found set: 3</td>
<td></td>
</tr>
</tbody>
</table>

**CDML tag name: Current Repeat Number**

Tag is replaced with the repetition currently being processed. This is always found within a [FMP-Repeating][/FMP-Repeating] loop.

**CDML syntax:** [FMP-CurrentRepeatNumber]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{position()}`
- otherwise: `<xsl:value-of select="position()"/>`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[FMP-Repeating: extensions] [FMP-CurrentRepeatNumber]: [FMP-RepeatingItem]&lt;br&gt;[/FMP-Repeating]</td>
<td></td>
</tr>
</tbody>
</table>

```
**Conversion example**

**Converted XSLT-CWP:**

```xml
<xsl:for-each select="fmrs:field[@name = 'extensions']/fmrs:data">
  <xsl:value-of select="position()" />
  <xsl:value-of select="." />
  <br />
</xsl:for-each>
```

**Converted Result:**

3: Green<br>

---

**CDML tag name: Current Skip Setting**

Tag is replaced with the number of records skipped from the beginning of the found set.

**CDML syntax:** [FMP-CurrentSkip]

**XSLT-CWP conversion:** `<xsl:call-template name="$get-current-skip" />`

- when inside SGML element attribute value: `{$current-skip}`
- otherwise: `<xsl:value-of select="$current-skip" />`
- if used in document, `$current-skip` variable is created at top-level using named template from the `cdml2xsl_utilities.xsl` stylesheet.

**Conversion example**

**Original CDML:** Current Skip Value is: [FMP-CurrentSkip]

**Converted XSLT-CWP:** Current Skip Value is: `<xsl:value-of select="$current-skip" />`

**Converted Result:** Current Skip Value is: 10

---

**CDML tag name: Current Sort Order**

Repeats the HTML between [FMP-CurrentSort] and [/FMP-CurrentSort] for each sort argument that was part of the request that created this page.

**CDML syntax:** [FMP-CurrentSort][...][/FMP-CurrentSort]

**XSLT-CWP conversion:**

- `<xsl:for-each select="$current-sort/sort-field">...</xsl:for-each>`
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-CurrentSort] not in a valid location -->
- if used in document, `$current-sort` variable is created at top-level using named template from the `cdml2xsl_utilities.xsl` stylesheet.

**Conversion example**

**Original CDML:**

Current sort order is:<br>
[FMP-CurrentSort]<br>
Field: [FMP-SortFieldItem], Order: [FMP-SortOrderItem]<br>
[/FMP-CurrentSort]

**Converted XSLT-CWP:**

Current sort order is:<br>

`<xsl:for-each select="$current-sort/sort-field">`<br>
Field: `<xsl:value-of select="@name" />`, Order: `<xsl:value-of select="@order" />`<br>
`<br /`<br>
`</xsl:for-each>`

---
CDML tag name: Current Time
Tag is replaced with the current time
CDML syntax: [FMP-CurrentTime: Format], where Format is 'Short' (default), or 'Long'
XSLT-CWP conversion:
- when inside SGML element attribute value, 'Short': {fmxslt:get_time('short')}
- or, 'Long': {fmxslt:get_time('long')}
- otherwise, 'Short': <xsl:value-of select="fmxslt:get_time('short')" />
- or, 'Long': <xsl:value-of select="fmxslt:get_time('long')" />

Conversion example

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current time is: [FMP-CurrentTime: Short]</td>
<td>Current time is: &lt;xsl:value-of select=&quot;fmxslt:get_time('short')&quot; /&gt;</td>
<td>Current time is: 10:12 AM</td>
</tr>
</tbody>
</table>

CDML tag name: Current Token
Tag is replaced with the value of -Token variable tag used to create the current page
CDML syntax: [FMP-CurrentToken: Number, Encoding], where Number is '0'...'9' (optional), and Encoding is 'Raw', 'URL', or 'HTML' (default)
XSLT-CWP conversion:
- when inside SGML element attribute value: {$request-query/fmq:query/fmq:parameter[@NAME='-token']}
- otherwise: <xsl:value-of select="$request-query/fmq:query/fmq:parameter[@NAME='-token']" />
- if the token has a Number, the query parameter is '-token.Number'; for example, '-token.5

Conversion example

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>The value of token.5 is [FMP-CurrentToken: 5, HTML]</td>
<td>The value of token.5 is &lt;xsl:value-of select=&quot;$request-query/fmq:query/fmq:parameter[@NAME='-token.5']&quot; /&gt;</td>
<td>The value of token.5 is MyValue</td>
</tr>
</tbody>
</table>
**CDML tag name: Else**

Tag is replaced with the specified data if the preceding [FMP-If] BooleanExpression is false

**CDML syntax:** `[FMP-If: BooleanExpression1]...[FMP-Else]...[/FMP-If]`

**XSLT-CWP conversion:**
- `<xsl:choose><xsl:when test="BooleanExpression1">...</xsl:when><xsl:otherwise>...</xsl:otherwise></xsl:choose>`
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-Else] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States of America</td>
<td><code>Other country</code></td>
</tr>
<tr>
<td>[FMP-Else]</td>
<td></td>
</tr>
<tr>
<td>Other country</td>
<td></td>
</tr>
<tr>
<td>[FMP-If]</td>
<td></td>
</tr>
</tbody>
</table>

**Converted Result:** Other country

---

**CDML tag name: Else If**

Tag is replaced with the specified data if BooleanExpression is true, and the preceding [FMP-If] BooleanExpression is false

**CDML syntax:** `[FMP-If: BooleanExpression1]...[FMP-ElseIf BooleanExpression2]...[/FMP-If]`

**XSLT-CWP conversion:**
- `<xsl:choose><xsl:when test="BooleanExpression1">... </xsl:when><xsl:when test="BooleanExpression2">... </xsl:when></xsl:choose>`
- BooleanExpression2 is converted to XPath
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-ElseIf] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
</tr>
</thead>
<tbody>
<tr>
<td>United States of America</td>
<td><code>Italy</code></td>
</tr>
<tr>
<td>[FMP-ElseIf: Field:country .Eq. Italy]</td>
<td></td>
</tr>
<tr>
<td>Italy</td>
<td></td>
</tr>
<tr>
<td>[FMP-If]</td>
<td></td>
</tr>
</tbody>
</table>
Conversion example

**CDML tag name: Field**

Tag is replaced with the contents of the specified field

**CDML syntax:** [FMP-Field: FieldName, Encoding], where Encoding is ‘Raw’, 'URL', 'HTML' (default), or 'Break'

**XSLT-CWP conversion:**

- when inside SGML element attribute value:
  - when current context is a record: \(<\text{xsl:when test="fmrs:field[@name = 'country']/fmrs:data[1] = 'us' or fmrs:field[@name = 'country']/fmrs:data[1] = 'usa'">United States of America</xsl:when>\)
  - when current context is a record: \(<\text{xsl:when test="fmrs:field[@name = 'country']/fmrs:data[1] = 'Italy'">Italy</xsl:when>\)

**Conversion example**

**Original CDML:** First Name: [FMP-Field: First Name]

**Converted XSLT-CWP:** First Name: \(<\text{xsl:value-of select="fmrs:field[@name = 'First Name']/fmrs:data[1]" />\)

**Converted Result:** First Name: John
**CDML tag name: Field Name**

Tag is replaced with the name of the current field. This is always found within a [FMP-LayoutFields] loop.

**CDML syntax:** [FMP-FieldName: Encoding], where Encoding is 'Raw', 'URL', or 'HTML' (default)

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{@name}`
- otherwise: `<xsl:value-of select="@name" />

**Conversion example**

**Original CDML:**
```
<select name="-SortField">
  <option value="">-None-</option>
  [FMP-LayoutFields]
  <option>[FMP-FieldName: Raw]</option>
  [/FMP-LayoutFields]
</select>
```

**Converted XSLT-CWP:**
```
<select name="-SortField">
  <option value=""/>-None-
  <xsl:for-each select="fmrs:fmresultset/fmrs:metadata/fmrs:field-definition" />
  <option><xsl:value-of select="@name"/> </option>
</xsl:for-each>
</select>
```

**Converted Result:**
```
<select name="-SortField">
  <option value="">-None-</option>
  <option>First Name</option>
  <option>Last Name</option>
  <option>Employee Number</option>
</select>
```
**CDML tag name: Find Field Item**

Tag is replaced with the field name that was part of the find request that created this page. This is always found within a [FMP-CurrentFind] loop.

**CDML syntax:** [FMP-FindFieldItem: *Encoding*], where *Encoding* is 'Raw', 'URL', or 'HTML' (default)

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{@name}`
- otherwise: `<xsl:value-of select="@name" />`
- if used in document, $current-find variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet
- when outside of a [FMP-CurrentFind] loop: <!-- CDML Converter ERROR: [FMP-FindFieldItem] outside of [FMP-CurrentFind] -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
</table>
| Current find request is:  
[FMP-CurrentFind]  
Field: [FMP-FindFieldItem], Op: [FMP-FindOpItem], Value: [FMP-FindValueItem]<br>[/FMP-CurrentFind] | Current find request is:<br/>  
<xsl:for-each select="xalan:nodeset($current-find)/find-field">  
</xsl:for-each> | Current find request is:  
Field: *First Name*, Op: begins with, Value: John<br>  
Field: *Last Name*, Op: equals, Value: Doe<br>
**CDML tag name: Find Operator Item**

Tag is replaced with the search operator that was part of the find request that created this page. This is always found within a [FMP-CurrentFind] loop.

**CDML syntax:** [FMP-FindOpItem: Format], where Format is 'Short', 'Long' (default), or 'Display'

**XSLT-CWP conversion:**
- when inside SGML element attribute value: { @short-operator } or { @long-operator }
- otherwise: <xsl:value-of select="@short-operator" /> or <xsl:value-of select="@long-operator" />
- if Format is 'Display' then 'Long' is used, and <!-- CDML2XSLT WARNING: [FMP-SortOrderItem] 'Display' Format not supported by XSLT-CWP -->
- if used in document, $current-find variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current find request is:</td>
<td>Current find request is:</td>
<td>Current find request is:</td>
</tr>
</tbody>
</table>
**CDML tag name: Find Value Item**
Tag is replaced with the value that was part of the find request that created this page. This is always found within a [FMP-CurrentFind] loop.

**CDML syntax**: [FMP-FindValueItem: Encoding], where Encoding is 'Raw', 'URL', or 'HTML' (default)

**XSLT-CWP conversion**:
- when inside SGML element attribute value: `{text()}
- otherwise: `<xsl:value-of select="text()" />
- if used in document, $current-find variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet
- when outside of a [FMP-CurrentFind] loop: `<!-- CDML Converter ERROR: [FMP-FindValueItem] outside of [FMP-CurrentFind] -->`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current find request is:</td>
<td>Current find request is:&lt;br /&gt;</td>
<td>Current find request is:</td>
</tr>
<tr>
<td>[FMP-CurrentFind]</td>
<td>&lt;xsl:for-each select=&quot;xalan:nodeset($current-find)/find-field&quot;&gt;</td>
<td>Field: First Name, Op: begins with, Value: John&lt;br&gt;</td>
</tr>
<tr>
<td>[FMP-CurrentFind]</td>
<td>&lt;/xsl:for-each&gt;</td>
<td></td>
</tr>
</tbody>
</table>
**CDML tag name: Header**

The HTTP header of the page sent to the browser is replaced with the text between the [FMP-Header] and [/FMP-Header] tags. None of the text between the tags will appear in the HTML portion of the page.

**CDML syntax:** [FMP-Header]...[/FMP-Header]

**XSLT-CWP conversion:**
- if a status code is in the header contents: `<xsl:variable name="header-status-code1" select="set_status_code('<status code from contents>')" />
- for each name-value pair in the header contents: `<xsl:variable name="header-param NUMBER" select="fmxslt:set_header('<param name>', '<param value>')" />
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-Header] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
</table>
| Header code is here:  
[FMP-Header]  
HTTP/1.0 302 Moved Temporary  
Location: http://www.FileMaker.com  
[/FMP-Header]  
| `<xsl:variable name="header-status-code1" select="set_status_code(302)" />
` | (The HTTP response also contains Location: http://www.filemaker.com in the header.) |

**CDML tag name: If**

Along with the [FMP-Else], [FMP-ElseIf] and [/FMP-If] tags, the [FMP-If] tag controls what HTML is shown by the browser.

**CDML syntax:** [FMP-If BooleanExpression]...[/FMP-If]

**XSLT-CWP conversion:**
- `<xsl:choose><xsl:when test="BooleanExpression">...
<xsl:when></xsl:choose>
  
  BooleanExpression is converted to XPath  
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-If] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
<th>Converted Result</th>
</tr>
</thead>
</table>
| [FMP-If: First Name .eq. field: Nick Name]  
Your nick name is the same as your name.  
[/FMP-If]  
| `<xsl:choose><xsl:when test="'First Name' = fmrs:field[@name = 'Nick Name']/fmrs:data[1]">  
Your nickname is the same as your name.  
</xsl:when></xsl:choose>` | Your nick name is the same as your name. |
**CDML tag name: Image**

Tag is replaced with a container URL reference to the image in the specified field

**CDML syntax:** [FMP-Image: FieldName]

**XSLT-CWP conversion:**

- when current context is a record: `<xsl:call-template name="get-image"><xsl:with-param name="fieldname" select="'fieldname'"/><xsl:with-param name="recid" select="'$record/@record-id'"/></xsl:call-template>`
- otherwise: `<xsl:call-template name="get-image"><xsl:with-param name="fieldname" select="'fieldname'"/><xsl:with-param name="recid" select="'$default-record/@record-id'"/></xsl:call-template>`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
</table>

**CDML tag name: Include**

Tag is replaced with the contents of another file, usually an HTML format file

**CDML syntax:** [FMP-Include: FileName]

**XSLT-CWP conversion:**

- `<!-- CDML Converter WARNING: The file “FileName” has now been incorporated inline into this document and the reference to the original file has been dropped; the original file may no longer be needed. -->converted contents of the file`
- if inside an SGML tag/attribute, `<!-- CDML Converter ERROR: [FMP-Include] not in a valid location -->`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>[FMP-Include: requirefield.htm]</code></td>
<td><code>&lt;!-- CDML Converter WARNING: The file &quot;requirefield.htm&quot; has now been incorporated inline into this document and the reference to the original file has been dropped; the original file may no longer be needed. --&gt;converted contents of the file</code></td>
<td>converted contents of the file</td>
</tr>
</tbody>
</table>
**CDML tag name: Include Field**

Tag is replaced with the contents of a field, usually a text field that contains an HTML format file.

**CDML syntax:** [FMP-IncludeField: FieldName]

**XSLT-CWP conversion:**
- <!-- CDML Converter ERROR: [FMP-IncludeField] not supported by XSLT-CWP -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
</tr>
</thead>
<tbody>
<tr>
<td>[FMP-IncludeField: errorPage]</td>
<td>&lt;!-- CDML Converter ERROR: [FMP-IncludeField] not supported by XSLT-CWP --&gt;</td>
</tr>
</tbody>
</table>

**CDML tag name: Inline Action**

Tag allows the processing of multiple CDML requests during the processing of a single format file. The [FMP-InlineAction] tag takes as its parameters the URL-like format of the name value pairs for a CDML request. All further processing of the format file then continues as if the inline request started the processing.

**CDML syntax:** [FMP-InlineAction: intratags][FMP-InlineAction]

**XSLT-CWP conversion:**
- if there is non-white space content between the start and end tags, loop over the response document’s records
- the request URL parameter values are URL-encoded
- the request URL is created from the intratags ignoring any '-format' parameter
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-InlineAction] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
</tr>
</thead>
</table>
**Conversion example**

<table>
<thead>
<tr>
<th>Converted Result:</th>
<th>Title: Mona Lisa, Artist: Leonardo da Vinci&lt;br&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>Title: Sunflowers, Artist: Vincent Van Gogh&lt;br&gt;</td>
<td></td>
</tr>
</tbody>
</table>

### Custom Web Publishing Guide

**CDML tag name: Layout Fields**

Repeats the HTML between `[FMP-LayoutFields]` and `[/FMP-LayoutFields]` for all the fields on the layout specified that was part of the request that created this page.

**CDML syntax:** `[FMP-LayoutFields]...[/FMP-LayoutFields]`

**XSLT-CWP conversion:**

- `<xsl:for-each select="fmrs:fmresultset/fmrs:metadata/fmrs:field-definition" />...<xsl:for-each>`
- if inside an SGML tag/attribute, <!-- CDML Converter ERROR: [FMP-LayoutFields] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th><code>&lt;select name=&quot;-SortField&quot;&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;option value=&quot;&quot;&gt;-None-</code></td>
</tr>
<tr>
<td></td>
<td><code>[FMP-LayoutFields]</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;option&gt;[FMP-FieldName: Raw]</code></td>
</tr>
<tr>
<td></td>
<td><code>[/FMP-LayoutFields]</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/select&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Converted XSLT-CWP:</th>
<th><code>&lt;select name=&quot;-SortField.1&quot;&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;option value=&quot;&quot;&gt;-None-&lt;/option&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;xsl:for-each select=&quot;fmrs:fmresultset/fmrs:metadata/fmrs:field-definition&quot; /&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;option /&gt;&lt;xsl:value-of select=&quot;@name&quot; /&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/xsl:for-each&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/select&gt;</code></td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Converted Result:</th>
<th><code>&lt;select name=&quot;-SortField.1&quot;&gt;</code></th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td><code>&lt;option value=&quot;&quot;&gt;-None-</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;option&gt;First Name&lt;/option&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;option&gt;Last Name&lt;/option&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;option&gt;Employee Number&lt;/option&gt;</code></td>
</tr>
<tr>
<td></td>
<td><code>&lt;/select&gt;</code></td>
</tr>
</tbody>
</table>
**CDML tag name: Link**

Tag is replaced with a URL that refers to the page it is on. The parameters can be used to remove portions of the generated URL.

**CDML syntax:** `<sgml_tag sgml_attr="[FMP-Link: CharacterCodes]&-format=FileName.htm&name1=value1&name2=value2...">`

**XSLT-CWP conversion:**

- insert: `<xsl:call-template name="get-link"><xsl:with-param name="filter-codes" select="'CharacterCodes'"/><xsl:with-param name="stylesheet" select="'FileName.xsl'"/><xsl:with-param name="other-params" select="'&name1=value1&name2=value2...'"/></xsl:call-template>`

- if inside an SGML attribute and followed a '-format' parameter: the 'stylesheet' parameter is passed to the 'get-link' template

- if inside an SGML attribute and followed by extra query parameters: the 'other-params' parameter is passed to the 'get-link' template

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th><code>&lt;a href=&quot;[FMP-Link: adr]&amp;-Format=AltHitList.htm&amp;-db=art.fp5&amp;-FindAll=&quot;&gt;Alternate hit list&lt;/a&gt;</code></th>
</tr>
</thead>
</table>
| Converted XSLT-CWP: | `<a>
  <xsl:attribute name="href">
    <xsl:call-template name="get-link">
      <xsl:with-param name="filter-codes" select="'adr'"/>
      <xsl:with-param name="stylesheet" select="'AltHitList.xsl'"/>
      <xsl:with-param name="other-params" select="'&amp;-db=art'"/>
    </xsl:call-template>
  </xsl:attribute>
  Alternate hit list
</a>` |
| Converted Result: | `<a href="/fmi/xsl/AltHitList.xsl?-db=art&-grammar=fmresultset&-findAll=">Alternate hit list</a>` |
**CDML tag name: Link First**
The HTML between [FMP-LinkFirst] and [/FMP-LinkFirst] is replaced with a link to the first range of records based on the -Max value used to create the current page.

**CDML syntax:** [FMP-LinkFirst]...[/FMP-LinkFirst]

**XSLT-CWP conversion:**
- `<a href="{$link-first}">...</a>`
- insert at top-level of document: `<xsl:variable name="link-first">.<xsl:call-template name="get-link-first"/>
</xsl:variable>`
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-LinkFirst] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>[FMP-LinkFirst]First set of records[/FMP-LinkFirst]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted XSLT-CWP:</td>
<td><code>&lt;a href=&quot;{$link-first}&quot;&quot;&gt;First set of records&lt;/a&gt;</code></td>
</tr>
<tr>
<td>Converted Result:</td>
<td><code>&lt;a href=&quot;/fmi/xsl/stylesheet.xsl?-max=10&amp;-find=&quot;&quot;&gt;First set of records&lt;/a&gt;</code></td>
</tr>
</tbody>
</table>

**CDML tag name: Link Last**
The HTML between [FMP-LinkLast] and [/FMP-LinkLast] is replaced with a link to the last range of records based on the -Max value used to create the current page.

**CDML syntax:** [FMP-LinkLast]...[/FMP-LinkLast]

**XSLT-CWP conversion:**
- `<a href="{$link-last}"">...</a>`
- insert at top-level of document: `<xsl:variable name="link-last">.<xsl:call-template name="get-link-last"/>
</xsl:variable>`
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-LinkLast] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>[FMP-LinkLast]Last set of records[/FMP-LinkLast]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted XSLT-CWP:</td>
<td><code>&lt;a href=&quot;{$link-last}&quot;&quot;&gt;Last set of records&lt;/a&gt;</code></td>
</tr>
<tr>
<td>Converted Result:</td>
<td><code>&lt;a href=&quot;/fmi/xsl/stylesheet.xsl?-skip=10&amp;-max=10&amp;-find=&quot;&quot;&gt;Last set of records&lt;/a&gt;</code></td>
</tr>
</tbody>
</table>
**CDML tag name: Link Next**

The HTML between [FMP-LinkNext] and [/FMP-LinkNext] is replaced with a link to the next range of records based on the -Max and -Skip values used to create the current page.

**CDML syntax:** [FMP-LinkNext]...[/FMP-LinkNext]

**XSLT-CWP conversion:**
- `<a href="{$link-next}">...</a>
- insert at top-level of document: `<xsl:variable name="link-next"><xsl:call-template name="get-link-next"/>
</xsl:variable>
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-LinkNext] not in a valid location -->

**Conversion example**

| Original CDML: | [FMP-LinkNext]Next set of records[/FMP-LinkNext] |
|===============|-----------------------------------------------|
| Converted XSLT-CWP: | `<a href="{$link-next}">Next set of records</a>` |
| Converted Result: | `<a href="/fmi/xsl/stylesheet.xsl?skip=10&max=10&find=">Next set of records</a>` |

**CDML tag name: Link Previous**

The HTML between [FMP-LinkPrevious] and [/FMP-LinkPrevious] is replaced with a link to the previous range records based on the -Max and -Skip value used to create the current page.

**CDML syntax:** [FMP-LinkPrevious]...[/FMP-LinkPrevious]

**XSLT-CWP conversion:**
- `<a href="{$link-previous}">...</a>
- insert at top-level of document: `<xsl:variable name="link-previous"><xsl:call-template name="get-link-previous"/>
</xsl:variable>
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-LinkPrevious] not in a valid location -->

**Conversion example**

| Original CDML: | [FMP-LinkPrevious]Previous set of records[/FMP-LinkPrevious] |
|===============|-----------------------------------------------|
| Converted XSLT-CWP: | `<a href="{$link-previous}">Previous set of records</a>` |
| Converted Result: | `<a href="/fmi/xsl/stylesheet.xsl?skip=10&max=10&find=">Previous set of records</a>` |
**CDML tag name: Link to Record ID**

Tag is replaced with a URL to a specific record in a database. The link that is generated includes all the search and sort criteria that was used to generate the page this tag was on. The Layout parameter is optional.

**CDML syntax:** [FMP-LinkRecID: Format=FileName, Layout=LayoutName]

**XSLT-CWP conversion:**
- `<xsl:call-template name="get-link-rec-id">`<xsl:with-param name="rec-id" select="@record-id"/>
  `<xsl:with-param name="stylesheet" select=""FileNamel.xsll""/></xsl:call-template>`
- if followed a '-format' parameter: the 'stylesheet' parameter is passed to the 'get-link-rec-id' template
- if followed a '-lay' parameter: the 'layout' parameter is passed to the 'get-link-rec-id' template
- if not inside an SGML attribute: <!-- CDML Converter ERROR: [FMP-LinkRecID] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;a href=&quot;[FMP-LinkRecID: format=FormatFile.htm, layout=detail]&quot;&gt;More detail&lt;/a&gt;</code></td>
<td><code>&lt;a href=&quot;/fmi/xsl/FormatFile.xsl?-db=products&amp;-grammar=fmresultset&amp;-lay=detail&amp;-recid=3&amp;-find&quot;&gt;More detail&lt;/a&gt;</code></td>
</tr>
</tbody>
</table>

**CDML tag name: Log**

Tag is replaced with the text specified as the parameter and written as user log messages in the application log file. See “Using the Web Publishing Engine application log” on page 73.

**CDML syntax:** [FMP-Log: someText]

**XSLT-CWP conversion:**
- `<xsl:message>`someText</xsl:message>
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-Log] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Log code is here: [FMP-Log: the search page was accessed]</td>
<td>Log code is here: <code>&lt;xsl:message&gt;</code>the search page was accessed&lt;/xsl:message&gt;`</td>
<td>Log code is here: (The text the search page was accessed is written to the application log if user logging is enabled.)</td>
</tr>
</tbody>
</table>
**CDML tag name: Option**

Tag replaced with all the values in the value list of the field in that layout

**CDML syntax:** `[FMP-Option: FieldName, List=ValueListName]`

**Note** To be converted properly, the "[FMP-Option]" CDML tag requires field(s) on the layout that reference the specified value list(s).

**XSLT-CWP conversion:**

- `<xsl:variable name="valuelist-name" select="$/layout/fml:FMPXMLLAYOUT/fml:LAYOUT/fml:FIELD[@NAME = 'FieldName']/fml:STYLE/@VALUELIST"/>`  
  `<xsl:for-each select="$/layout/fml:FMPXMLLAYOUT/fml:VALUELISTS/fml:VALUELIST[@NAME = $valuelist-name]/fml:VALUE">`  
  `<xsl:value-of select="current()"/>`  
  `<xsl:if test="current() = $default-record/fmrs:field[@name = 'FieldName']/fmrs:data[1]">`  
  `<xsl:attribute name="selected">selected</xsl:attribute></xsl:if>`

- when current context is a record: use `fmrs:field[@name = 'FieldName']/fmrs:data[1]` above

- parameter 'List' is ignored: <!-- CDML2XSLT WARNING: [FMP-Option] 'List' parameter not supported by XSLT-CWP -->
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-Option] not in a valid location -->
- if used in document: $layout variable is created at top-level of document

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
</tr>
</thead>
</table>
| `<select name="Groups">[FMP-option: Groups]</select>` | `<select>`  
  `<xsl:variable name="valuelist-name" select="$/layout/fml:FMPXMLLAYOUT/fml:LAYOUT/fml:FIELD[@NAME = 'Groups']/fml:STYLE/@VALUELIST"/>`  
  `<xsl:for-each select="$/layout/fml:FMPXMLLAYOUT/fml:VALUELISTS/fml:VALUELIST[@NAME = $valuelist-name]/fml:VALUE">`  
  `<option value="{current()}"/>`  
  `<xsl:if test="current() = $default-record/fmrs:field[@name = 'Groups']/fmrs:data[1]">`  
  `<xsl:attribute name="selected">selected</xsl:attribute></xsl:if>`  
  `<xsl:value-of select="current()"/>`  
  `<xsl:if test="current() = $default-record/fmrs:field[@name = 'Groups']/fmrs:data[1]">`  
  `<xsl:attribute name="selected">selected</xsl:attribute></xsl:if>`

| Converted Result: | `<select name="Groups">`  
  `<option>Production</option>`  
  `<option selected>Support</option>`  
  `<option>Sales</option></select>` |
**CDML tag name: Portal**
Repeats the HTML between [FMP-Portal] and [/FMP-Portal] for each record in a specified portal

**CDML syntax:** [FMP-Portal: RelationshipName]...[/FMP-Portal]

**XSLT-CWP conversion:**
- when current context is a record: `<xsl:for-each select="fmrs:relatedset[@table = 'RelationshipName']/fmrs:record">...</xsl:for-each>`
- otherwise: `<xsl:for-each select="/fmrs:fmresultset/fmrs:resultset/fmrs:record/fmrs:relatedset[@table='RelationshipName']/fmrs:record">...</xsl:for-each>`
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-Portal] not in a valid location -->

**Conversion example**

| Original CDML: | [FMP-Portal: lineitems]
|               | [FMP-CurrentPortalRowNumber]: [FMP-Field: lineitems::name]<br>
|               | [/FMP-Portal] |
| Converted XSLT-CWP: | `<xsl:for-each select="fmrs:relatedset[@table='lineitems']/fmrs:record">`
|               | `<xsl:value-of select="position()"/>: <xsl:value-of select="fmrs:field[@name = 'name']" /><br />
|               | `</xsl:for-each>` |
| Converted Result: | 1: Red<br>|

**CDML tag name: Range End**
Tag is replaced with the record number of the last record being displayed

**CDML syntax:** [FMP-RangeEnd]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{current-skip + /fmrs:fmresultset/fmrs:resultset/@fetch-size}`
- if used in document: $current-skip variable is created at top-level using named template from the cdml2xslUtilities.xsl stylesheet

**Conversion example**

| Original CDML: | Records [FMP-RangeStart] through [FMP-RangeEnd] |
| Converted Result: | Records 6 through 10 |
**CDML tag name: Range Size**

Tag is replaced with the number of records displayed on the page

**CDML syntax:** [FMP-RangeSize]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{/fmrs:fmresultset/fmrs:resultset/@fetch-size}`

### Conversion example

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>You are viewing [FMP-RangeSize] records.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted Result:</td>
<td>You are viewing 8 records.</td>
</tr>
</tbody>
</table>

**CDML tag name: Range Start**

Tag is replaced with the record number of the first record being displayed

**CDML syntax:** [FMP-RangeStart]

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{$current-skip + 1}`
- otherwise: `<xsl:value-of select="$current-skip + 1"/>`
- if used in document: `$current-skip` variable is created at top-level using named template from the `cdml2xsl_utilities.xsl` stylesheet

### Conversion example

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Records [FMP-RangeStart] through [FMP-RangeEnd]</th>
</tr>
</thead>
<tbody>
<tr>
<td>Converted Result:</td>
<td>Records 6 through 10</td>
</tr>
</tbody>
</table>
**CDML tag name: Record**

Repeats the HTML between [FMP-Record] and [/FMP-Record] for every record up to -Max, starting after skipping -Skip records

CDML syntax: [FMP-Record]...[/FMP-Record]

XSLT-CWP conversion:
- `<xsl:for-each select="/fmrs:fmresultset/fmrs:resultset/fmrs:record">...</xsl:for-each>`
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-Record] not in a valid location -->

**Conversion example**

| Original CDML: | [FMP-Record]  
|               | [FMP-Field: Country] - [FMP-Field: Capital]<br>  
|               | [/FMP-Record]  
| Converted XSLT-CWP: | `<xsl:for-each select="/fmrs:fmresultset/fmrs:resultset/fmrs:record">`  
|              | `<xsl:value-of select="fmrs:field[@name = 'Country']/fmrs:data[1]" /> - `  
|              | `<xsl:value-of select="fmrs:field[@name = 'Capital']/fmrs:data[1]" />`  
|              | `<br />`  
|              | `</xsl:for-each>`  
| Converted Result: | Great Britain - London<br>  
|                  | France - Paris<br>  
|                  | USA - Washington D.C.<br>  

**CDML tag name: Repeating**

Repeats the HTML between [FMP-Repeating] and [/FMP-Repeating] for all the repetitions of the specified field. The tag [FMP-RepeatingItem] is replaced with the contents of a specific repetition.

CDML syntax: [FMP-Repeating: FieldName]...[/FMP-Repeating]

XSLT-CWP conversion:
- when current context is a record: `<xsl:for-each select="fmrs:field[@name = 'FieldName']/fmrs:data">...</xsl:for-each>`
- otherwise: `<xsl:for-each select="/fmrs:fmresultset/fmrs:resultset/fmrs:record[1]/fmrs:field[@name = 'FieldName']/fmrs:data">...</xsl:for-each>`
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-Repeating] not in a valid location -->

**Conversion example**

| Original CDML: | [FMP-Repeating: extensions]  
|               | [FMP-CurrentRepeatNumber]: [FMP-RepeatingItem]<br>  
|               | [/FMP-Repeating]  
| Converted XSLT-CWP: | `<xsl:for-each select="fmrs:field[@name = 'extensions']/fmrs:data">`  
|                  | `<xsl:value-of select="position()" />: <xsl:value-of select="." />`<br>  
|                  | `</xsl:for-each>`  
| Converted Result: | 3: Green<br>
**CDML tag name: Repeating Item**

Tag is replaced with the contents of the next repetition. This is always found within a [FMP-Repeating] loop.

**CDML syntax:** [FMP-RepeatingItem: Encoding], where Encoding is 'Raw', 'URL', 'HTML' (default), or 'Break'

**XSLT-CWP conversion:**
- when inside SGML element attribute value: {.}
- otherwise: `<xsl:value-of select="." />`

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

**CDML tag name: Set Cookie**

Tag is replaced with nothing in the HTML and is used to store a variable in the user’s browser. The Expires, Path, and Domain parameters are optional.

**CDML syntax:** [FMP-SetCookie: name=value, Expires=expires, Path=path, Domain=domain]

**XSLT-CWP conversion:**
- `<xsl:variable name="cookie-name" select="fmxslt:set_cookie('name', 'value', 'expires', 'path', 'domain')" />`
- undefined optional parameters expires, path, domain are passed-as empty strings, ""
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-SetCookie] not in a valid location -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>![Image]</td>
<td>![Image]</td>
<td>![Image]</td>
</tr>
</tbody>
</table>

Conversion example

Original CDML: `[FMP-Repeating: extensions]
[FMP-CurrentRepeatNumber]: [FMP-RepeatingItem]<br>`

Converted XSLT-CWP: `<xsl:for-each select="fmrs:field[@name = 'extensions']/fmrs:data">
<xsl:value-of select="position()" />:
<xsl:value-of select="." /><br />
</xsl:for-each>`

Converted Result: `3: Green<br>`

Conversion example

Original CDML: `Set-Cookie here: [FMP-SetCookie: ColorChoice=Green, Expires=43200]`


Converted Result: `Set-Cookie here:`
**CDML tag name: Sort Field Item**

Tag is replaced with the field name that was part of the request that created this page. This is always found within a [FMP-CurrentSort] loop.

**CDML syntax**: `[FMP-SortFieldItem: Encoding], where Encoding is 'Raw', 'URL', or 'HTML' (default)``

**XSLT-CWP conversion**: 
- when inside SGML element attribute value: `{@name}`
- otherwise: `<xsl:value-of select="@name" />`
- if used in document: $current-sort variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet
- when outside of a [FMP-CurrentSort] loop: <!-- CDML Converter ERROR: [FMP-SortFieldItem] outside of [FMP-CurrentSort] -->

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td>Current sort order is:&lt;br&gt; [FMP-CurrentSort]&lt;br&gt; Field: [FMP-SortFieldItem], Order: [FMP-SortOrderItem]&lt;br&gt;[/FMP-CurrentSort]</td>
<td>&lt;xsl:for-each select=&quot;xalan:nodeset($current-find)/find-field&quot;&gt;&lt;br&gt; Field: &lt;xsl:value-of select=&quot;@name&quot; /&gt;, Order: &lt;xsl:value-of select=&quot;@order&quot; /&gt;&lt;br&gt;&lt;/xsl:for-each&gt;</td>
<td>Current sort order is:&lt;br&gt; Field: First Name, Order: descend&lt;br&gt; Field: Last Name, Order: descend</td>
</tr>
</tbody>
</table>

**CDML tag name: Sort Order Item**

Tag is replaced with the sort order that was part of the request that created the page. This is always found within a [FMP-CurrentSort] loop.

**CDML syntax**: `[FMP-SortOrderItem: Encoding], where Encoding is 'Raw', 'URL', 'HTML' (default), or 'Display'``

**XSLT-CWP conversion**: 
- when inside SGML element attribute value: `{@order}`
- otherwise: `<xsl:value-of select="@order" />`
- if Encoding is 'Display' then 'HTML' is used: <!-- CDML2XSLT WARNING: [FMP-SortOrderItem] 'Display' Encoding not supported by XSLT-CWP -->
if used in document: $current-sort variable is created at top-level using named template from the cdml2xsl_utilities.xsl stylesheet


Conversion example

| Original CDML: | Current sort order is:<br> | [FMP-CurrentSort] | Field: [FMP-SortFieldItem], Order: [FMP-SortOrderItem]<br> | [FMP-CurrentSort] |
| Converted XSLT-CWP: | Current sort order is:<br> | <xsl:for-each select="xalan:nodeset($current-find)/find-field"> | Field: <xsl:value-of select="@name" />, Order: <xsl:value-of select="@order" />/<br> | </xsl:for-each> |
| Converted Result: | Current sort order is:<br> | Field: First Name, Order: descend<br> | Field: Last Name, Order: descend<br> |

**CDML tag name: Value List**

Repeats the HTML between [FMP-ValueList] and [/FMP-ValueList] for all the values in the specified value list

**CDML syntax:** [FMP-ValueList: FieldName, List=ValueListName]...[/FMP-ValueList]

**Note** To be converted properly, the "[FMP-ValueList]" tag requires field(s) on the layout that reference the specified value list(s).

**XSLT-CWP conversion:**

- <xsl:variable name="valuelist-name" select="$layout/fml:FMPXMLLAYOUT/fml:LAYOUT/fml:FIELD[@NAME = 'FieldName']/fml:STYLE/@VALUELIST"/>...<xsl:for-each select="$layout/fml:FMPXMLLAYOUT/fml:VALUELISTS/fml:VALUELIST[@NAME = $valuelist-name]/fml:VALUE"/>...<xsl:for-each> parameter 'List' is ignored: <!-- CDML2XSLT WARNING: [FMP-ValueList] 'List' parameter not supported by XSLT-CWP -->
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-ValueList] not in a valid location -->
- if used in document: $layout variable is created at top-level of document

Conversion example

| Original CDML: | [FMP-ValueList: Groups, List=GroupList] | <input type="radio" name="Groups" value="[FMP-ValueListItem]" [FMP-ValueListChecked]>[FMP-ValueListItem] | [/FMP-ValueList] |
Conversion example

**Original CDML:**
```xml
[FMP-ValueList: Groups, List=GroupList]
<input type="radio" name="Groups" value="[FMP-ValueListItem]" [FMP-ValueListChecked]>[FMP-ValueListIterm]
[/FMP-ValueList]
```

**Converted XSLT-CWP:**
```xml
<xsl:variable name="valuelist-name" select="$layout/fml:FMPXMLLAYOUT/fml:LAYOUT/fml:FIELD[@NAME = 'Groups']/fml:STYLE/@VALUELIST"/>
  <xsl:for-each select="$layout/fml:FMPXMLLAYOUT/fml:VALUELISTS/fml:VALUELIST[@NAME = $valuelist-name]/fml:VALUE">
    <!-- CDML2XSLT WARNING: [FMP-ValueList] 'List' parameter not supported by XSLT-CWP -->
    <input type="radio" name="Groups" value="{current()}">{current()}</input>
    <xsl:if test="current() = fmrs:field[@name = 'Groups']/fmrs:data[1]">"checked"</xsl:if></xsl:if>
  </xsl:for-each>
```

**Converted Result:**
```
<input type="radio" name="Groups" value="Production">Production
<input type="radio" name="Groups" value="Sales" checked>Sales
<input type="radio" name="Groups" value="Support">Support
```

**CDML tag name: Value List Checked**

Tag is replaced with the word "checked" for every value list item that has been selected in the specified field. This is always found within a [FMP-ValueList] loop.

**CDML syntax:** `<input name="fieldname" value="fieldvalue" [FMP-ValueListChecked]>`

**XSLT-CWP conversion:**
- `<xsl:variable name="valuelist-name" select="$layout/fml:FMPXMLLAYOUT/fml:LAYOUT/fml:FIELD[@NAME = 'Groups']/fml:STYLE/@VALUELIST"/>
  <xsl:for-each select="$layout/fml:FMPXMLLAYOUT/fml:VALUELISTS/fml:VALUELIST[@NAME = $valuelist-name]/fml:VALUE">
    <!-- CDML2XSLT WARNING: [FMP-ValueList] 'List' parameter not supported by XSLT-CWP -->
    <input type="radio" name="Groups" value="{current()}">{current()}</input>
    <xsl:if test="current() = fmrs:field[@name = 'Groups']/fmrs:data[1]">"checked"</xsl:if></xsl:if>
  </xsl:for-each>`

- if not inside an SGML 'input' tag: `<!-- CDML Converter ERROR: [FMP-ValueListChecked] not in a valid location -->`
- if used in document: `$layout variable is created at top-level of document`
### Conversion example

**Converted Result:**

```xml
<input type="radio" name="Groups" value="Production">Production
<input type="radio" name="Groups" value="Sales" checked>Sales
<input type="radio" name="Groups" value="Support">Support
```

### CDML tag name: Value List Item

Tag is replaced with the next element of a value list. This is always found within a [FMP-ValueList] loop.

**CDML syntax:** [FMP-ValueListItem: Checked, Encoding], where Encoding is 'Raw', 'URL', or 'HTML' (default)

**XSLT-CWP conversion:**

- when inside SGML element attribute value: `{current()}`
- otherwise: `<xsl:value-of select="current()" />`
- if used in document: $layout variable is created at top-level of document

### Conversion example

<table>
<thead>
<tr>
<th>Original CDML</th>
<th>Converted XSLT-CWP</th>
</tr>
</thead>
</table>
| [FMP-ValueList: Groups, List=GroupList]  
<input type="radio" name="Groups" value="[FMP-ValueListItem]" [FMP-ValueListChecked]>  

<xsl:for-each select="$layout/fml:FMPXMLLAYOUT/fml:VALUELISTS/fml:VALUELIST[@NAME = $valuelist-name]/fml:VALUE">

<!-- CDML2XSLT WARNING: [FMP-ValueList] 'List' parameter not supported by XSLT-CWP -->

<input type="radio" name="Groups" value="{current()}"><xsl:if test=".= $current-record/fmrs:field[@name = 'Groups']/fmrs:data[1]"> <xsl:attribute name="checked">checked</xsl:attribute></xsl:if></input><xsl:value-of select="{current()}" />
```

**Converted Result:**

```xml
<input type="radio" name="Groups" value="Production">Production
<input type="radio" name="Groups" value="Sales" checked>Sales
<input type="radio" name="Groups" value="Support">Support
```
CDML tag name: Value Name Item

Tag is replaced by the name of a value list. This is always found within a [FMP-ValueNames] loop.

**CDML syntax:** [FMP-ValueNameItem: Encoding], where Encoding is 'Raw', 'URL', or 'HTML' (default)

**XSLT-CWP conversion:**
- when inside SGML element attribute value: `{@NAME}`
- otherwise: `<xsl:value-of select="@NAME" />`
- if inside an SGML tag/attribute: `<![CDATA[CDML Converter ERROR: [FMP-ValueNames] not in a valid location --]]>
- when outside of a [FMP-ValueNames] loop: `<![CDATA[CDML Converter ERROR: [FMP-ValueNameItem] outside of [FMP-ValueNames] --]]>
- if used in document: `$layout variable is created at top-level of document`

**Conversion example**

**Original CDML:**
```cdml
<select name="-sortorder">
  <option>Ascending</option>
  <option>Descending</option>
  [FMP-ValueNames]
  <option value="Custom=[FMP-ValueNameItem]">[FMP-ValueNameItem]</option>
  [/FMP-ValueNames]
</select>
```

**Converted XSLT-CWP:**
```xml
<select name="-sortorder.1">
  <option>Ascending</option>
  <option>Descending</option>
  <xsl:for-each select="$layout/fml:fmpxmllayout/fml:VALUELISTS/fml:VALUELIST">
    <option value="Custom={@NAME}" /> <xsl:value-of select="@NAME" /></option>
  </xsl:for-each>
</select>
```

**Converted Result:**
```xml
<select name="-sortorder.1">
  <option>Ascending</option>
  <option>Descending</option>
  <option value="Custom=Colors">Colors</option>
  <option value="Custom=Sizes">Sizes</option>
</select>
```
**CDML tag name: Value Names**
Repeats the HTML between [FMP-ValueNames] and [/FMP-ValueNames] for all the value lists that are in the database.

**CDML syntax:** [FMP-ValueNames]...[/FMP-ValueNames]

**Note** To be converted properly, the "[FMP-ValueNames]" CDML tag requires field(s) on the layout that reference the specified value list(s).

**XSLT-CWP conversion:**
- `<xsl:for-each select="$layout/fml:fmpxmlayout/fml:VALUELISTS/fml:VALUELIST">...</xsl:for-each>
- if inside an SGML tag/attribute: <!-- CDML Converter ERROR: [FMP-ValueNames] not in a valid location -->
- if used in document: $layout variable is created at top-level of document

**Conversion example**

<table>
<thead>
<tr>
<th>Original CDML:</th>
<th>Converted XSLT-CWP:</th>
<th>Converted Result:</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>&lt;select name=&quot;-sortorder&quot;&gt;</code></td>
<td><code>&lt;select name=&quot;-sortorder.1&quot;&gt;</code></td>
<td><code>&lt;select name=&quot;-sortorder.1&quot;&gt;</code></td>
</tr>
<tr>
<td><code>&lt;option&gt;Ascending</code></td>
<td><code>&lt;option /&gt;Ascending</code></td>
<td><code>&lt;option /&gt;Ascending</code></td>
</tr>
<tr>
<td><code>&lt;option&gt;Descending</code></td>
<td><code>&lt;option /&gt;Descending</code></td>
<td><code>&lt;option /&gt;Descending</code></td>
</tr>
<tr>
<td><code>&lt;option value=&quot;Custom=[FMP-ValueNameItem]&quot;&gt;[FMP-ValueNameItem]</code></td>
<td><code>&lt;option value=&quot;Custom={@NAME}&quot; /&gt;</code></td>
<td><code>&lt;option value=&quot;Custom={@NAME}&quot; /&gt;</code></td>
</tr>
<tr>
<td><code>[/FMP-ValueNames]</code></td>
<td><code>&lt;xsl:value-of select=&quot;@NAME&quot; /&gt;</code></td>
<td><code>&lt;xsl:value-of select=&quot;@NAME&quot; /&gt;</code></td>
</tr>
<tr>
<td><code>&lt;/select&gt;</code></td>
<td><code>&lt;/xsl:for-each&gt;</code></td>
<td><code>&lt;/xsl:for-each&gt;</code></td>
</tr>
</tbody>
</table>

```xml
<select name="-sortorder.1">
  <option />Ascending
  <option />Descending
  <xsl:for-each select="$layout/fml:fmpxmlayout/fml:VALUELISTS/fml:VALUELIST">
    <option value="Custom={@NAME}" />
    <xsl:value-of select="@NAME" />
  </xsl:for-each>
</select>
```
Index

A
access log files for web server, described 72
access privileges 18
accounts and privileges
  enabling for Custom Web Publishing 17
Guest account 18
scripts 13
Administration Console for Web Publishing Engine 16, 18, 22, 39
application log 70, 73
ASCII characters, in XML documents 33
authenticated base URI parameter 56
authentication of web users 17, 56, 57

B
base URI parameter 56
Basic Authentication for web users 17, 56
break_encode() extension function 64
buffering, using in stylesheet 58

C
CDML conversion to XSLT
  conversion errors, fixing 44, 108
described 99
  mapping rules 44, 99
obsolete action tags, conversion of 103
obsolete variable tags, conversion of 104
process of 99
  supported action tags, conversion of 104
  supported variable tags, conversion of 105
CDML Converter
described 11, 42, 99
generated stylesheets, fixing 44, 99
generated stylesheets, testing 45
generated stylesheets, using 45
installing 42
  starting and using 43
Change Password script 18
check_error_status() extension function 70, 97
checkboxes, checking for values in 65
client information, obtaining via XSLT parameters 56
client-side stylesheets 23, 36
commands for queries. See query strings
compare_date() extension function 67
cmpare_datetime() extension function 68
cmpare_day() extension function 67
cmpare_time() extension function 67
comparing strings 65
cmparison operators for fields 83
container fields
  how web users access data 19
publishing contents of 19, 45
URL syntax for accessing in XML solutions 24
URL syntax for accessing in XSLT solutions 48
contains_checkbox_value() extension function 65
content buffering, using 58
conversion log, CDML 44
cmpvert_datetime() extension function 68
cookies
  extension functions, using 63
  storing session ID 59
create_session() extension function 59
Custom Web Publishing
  access to solutions by web users 17, 56, 57
described 9, 11
  enabling in database 17
  enabling in Web Publishing Engine 18
  extended privilege for 17
Guest account 18
new features in 12
requirements for 12
restricting IP address access in web server 18
scripts 14
  using a static IP address 12
  using scripts 13
using XML 21
using XSLT 38, 47
Custom Web Publishing Engine (CWPE) 22, 38

D
database error codes 26
database sessions, enabling 60, 79
databases, protecting when published 18
date extension functions, using 66
date format strings 68
day extension functions, using 66
-db query parameter 82
-dbnames query command 79
-delete query command 79
Development mode, Web Publishing Engine 95
document type definitions (DTDs) 26, 27, 30
document() function 57
documentation information 10, 16
documents, loading via document() function 57
-dup query command 79

E
-edit query command 80
electronic documentation 10
Custom Web Publishing Guide

elements
  database error code 26
  in FMPXMLLAYOUT grammar 31
  in FMPXMLRESULT grammar 30
  in fmresultset grammar 27
email messages
  extension functions for 61
  initial default encoding setting 52
embedded forms 109
enabling Custom Web Publishing in database 17
  encoding
    -encoding query parameter 52, 82
    output via <xsl:output> element 53
    requests 52
    URLs 25, 59
  using string manipulation extension functions 64
  XML data 26, 33
  XSLT stylesheets 53
    -encoding query parameter 82
  <error code> and <ERRORCODE> elements 89
errors
  about error codes 89
  checking error status of extension functions 70, 97
  database error code elements 26
  database error code numbers 89
  extension function error code numbers 97
  log files for web server 72
  pe_application_log.txt log file 73
  pe_server_error.html log file 96
Web Publishing Engine error code numbers 95
examples of
  generated FMPXMLLAYOUT grammar 32
  generated FMPXMLRESULT grammar 30
  generated fmresultset grammar 28
export XML data 21
extended privilege for Custom Web Publishing 17
Extensible Markup Language (XML). See XML
Extensible Stylesheet Language Transformation (XSLT).
  See XSLT
Extension Function Reference for FileMaker XSLT 48
extension functions for FileMaker XSLT
  See also fmxslt extension functions

F
field name query parameter (non-container) 82
field names, fully qualified syntax 77
  -fieldname.op query parameter (container) 82
  -fieldname.op query parameter 83
FileMaker CDML Converter. See CDML Converter
FileMaker Pro, contrast with Web Publishing Engine 21
FileMaker Server, documentation information 10
FileMaker Site Assistant. See Site Assistant
FileMaker-specific XSLT parameters 54
  filtering data with stylesheets 37
  -find query command 80

FMPDSORESULT grammar 25
FMPXMLLAYOUT grammar 21, 25, 29, 32
FMPXMLRESULT grammar 21, 25, 29, 30
fmresultset grammar 21, 25, 27, 28
fmxml keyword for enabling XML publishing 17, 22
fmxslt extension functions
  fmxslt:break_encode() function 64
  fmxslt:check_error_status() function 70, 97
  fmxslt:compare_date() function 67
  fmxslt:compare_datetime() function 68
  fmxslt:compare_day() function 67
  fmxslt:compare_time() function 67
  fmxslt:contains_checkbox_value() function 65
  fmxslt:convert_datetime() function 68
  fmxslt:create_session() function 59
  fmxslt:get_cookie() function 63
  fmxslt:get_cookies() function 63
  fmxslt:get_date() function 67
  fmxslt:get_datetime() function 68
  fmxslt:get_day() function 67
  fmxslt:get_header() function 62
  fmxslt:get_long_date_format() function 67
  fmxslt:get_long_day_format() function 67
  fmxslt:get_long_time_format() function 67
  fmxslt:get_session_object() function 60
  fmxslt:get_short_date_format() function 67
  fmxslt:get_short_day_format() function 67
  fmxslt:get_short_time_format() function 67
  fmxslt:get_time() function 67
  fmxslt:html_encode() function 64
  fmxslt:invalidate_session() function 59, 60
  fmxslt:regex_contains() function 65
  fmxslt:remove_session_object() function 60
  fmxslt:send_email() functions 61
  fmxslt:session_encode_url() function 59
  fmxslt:session_exists() function 59
  fmxslt:set_cookie() function 63
  fmxslt:set_date() function 67
  fmxslt:set_datetime() function 68
  fmxslt:set_day() function 67
  fmxslt:set_session_object() function 60
  fmxslt:set_session_timeout() function 59
  fmxslt:set_status_code() function 62
  fmxslt:url_decode() function 64
  fmxslt:url_encode() function 64
fmxslt keyword for enabling XSLT publishing 17, 39
format strings, date and time 68
formatting data with stylesheets 37
  forms, nested 109
  fully qualified field name, syntax of 77

G
get_cookie() extension function 63
get_cookies() extension function 63
get_date() extension function 67
get_datetime() extension function 68
get_day() extension function 67
get_header() extension function 62
get_long_date_format() extension function 67
get_long_day_format() extension function 67
get_long_time_format() extension function 67
get_session_object() extension function 60
get_short_date_format() extension function 67
get_short_day_format() extension function 67
get_short_time_format() extension function 67
get_time() extension function 67
GIF files, publishing on web 19
global fields
database sessions, enabling 60, 79
syntax of 78
using with sessions 60, 79
-grammar query parameter 50, 84
grammars for XML, described 25
Guest account
disabling 18
enabling 18
with Custom Web Publishing 18

H
der header functions, using 62
hiding metadata with stylesheets 37
HTML
forms for XML requests 23
reformatting XML data into 21
html_encode() extension function 64

I
import XML data 21
Instant Web Publishing 9
integrating data with stylesheets 37
invalidate_session() extension function 59, 60
ISO-2022-JP encoding 52
ISO-8859-1 encoding 52
ISO-8859-15 encoding 52

J
JPEG files, publishing on web 19
jsessionid parameter 59

K
keywords for enabling Custom Web Publishing 17, 22, 39

L
-lay query parameter 35, 84
-lay.response query parameter 35, 84
layout information, using in stylesheet 57
-layoutnames query command 80
layouts, switching for an XML response 35
loading additional documents 57
log files 71, 73
described 72
logging via <xsl:message> element 70
pe_application_log.txt 73
pe_internal_access_log.txt 73
web server access 72
web_server_module_log.txt 73
<xsl:message> element 73
Logs folder 73
-lop query parameter 84

M
mail messages. See email messages
-max query parameter 85
metadata, hiding with stylesheets 37
method attribute, <xsl:output> element 53
migrating web publishing solutions 15
MIME (Multipurpose Internet Mail Extensions) types 18
-modid query parameter 85
monitoring web sites 72

N
namespaces for
XML 26
XSLT 50
nested forms 109
new features in Custom Web Publishing 12
-new query command 81
numbers for
database error codes 89
extension function error codes 97
Web Publishing Engine error codes 95

O
obsolete query request names and parameters 76
operators, comparison 83
order of XML request processing 35
output pages
coding, specifying 53
initial default encoding setting 52
output method, specifying 53
<xsl:output> element 53
outputting data with stylesheets 37
overview of steps for
XML data access 22
XSLT publishing 39

P
parameters for queries. See query strings
parameters for XSLT, FileMaker-specific 54
passwords
  access to XML documents 57
  Basic Authentication for web users 17, 56
  Change Password script 18
  defining for Custom Web Publishing 17
  no login password 18
pe_application_log.txt log file 73
pe_internal_access_log.txt log file 73
pe_server_error.html error page 96
Perl regular expressions, comparing strings 65
Portals
  adding records 77
  editing records 78
Privilege set, assigning for Custom Web Publishing 17
Production mode, Web Publishing Engine 96
Protecting published databases 18
Publishing on the web
  connecting to Internet or intranet 12
  container field objects 19, 45
  database error codes 89
  protecting databases 18
  QuickTime movies 19
  requirements for 12
  using Instant Web Publishing 9
  using XML 9, 11, 22
  using XSLT 9, 11, 39, 47
Query information, accessing in request 55
Query strings 33, 49, 75
  adding records to portals 77
  commands and parameters 33, 49, 75
  editing records in portals 78
  fully qualified field name, syntax of 77
  global fields, syntax of 78
  guidelines for 76
  obsolete request names and parameters 76
  Query Strings Reference 77
  requesting XML data 33, 75
  statically defined in XSLT stylesheets 51
  XSLT stylesheets, using in 49
Query Strings Reference 77
QuickTime movies, publishing on the web 19
R
-recid query parameter 85
regex_contains() extension function 65
Relogin script 18
remove_session_object() extension function 60
requests for XML data 23
requirements for Custom Web Publishing 12
S
Scalable Vector Graphics (SVG), transforming XML data into 21
-script query parameter 85
-script.prefind query parameter 86
-script.presort query parameter 86
-scriptnames query command 81
scripts
  accounts and privileges 13
  Change Password 18
  database sessions, enabling 60
  for XML requests 23
  in Custom Web Publishing 13
  Relogin 18
  tips and considerations 13
Security
  accounts and passwords 18
  FileMaker Security Guide 9
  guidelines for protecting published databases 18
  restricting access from IP addresses 18
  statically defined query strings, using 51
send_email() extension functions 61
server-side XSLT stylesheets 37, 47
session extension functions, using in stylesheets 58, 59
session_encode_url() extension function 59
session_exists() extension function 59
set_cookie() extension function 63
set_header() extension function 62
set_session_object() extension function 60
set_session_timeout() extension function 59
set_status_code() extension function 62
Shift_JIS encoding 52
Site Assistant
  described 11, 40
  generated stylesheets, described 41
  installing 40
  preparing for use 40
  starting 41
  using 41
-skip query parameter 86
-sortfield query parameter 86
-sortorder query parameter 87
SSL (Secure Sockets Layer) encryption 18
state, saving in sessions 58, 59
statically defined query strings in XSLT stylesheets 51
storing information in sessions 58, 59
strings
  comparing via Perl regular expressions 65
  using string manipulation extension functions 64
  stylehref query parameter 88
stylesheets
about 37
checkboxes, checking for values in 65
client-side 36
comparing strings via Perl regular expressions 65
content buffering, using 58
cookie extension functions 63
date and time format strings 68
date, time, and day extension functions 66
developing 47
email messages, sending 61
encoding of 53
error status of extension functions, checking 70
examples of usage 37
-grammar parameter for 50
guidelines for developing 47
header functions, using 62
query strings for 49
server-side 37, 47
session function, using 58, 59
string manipulation extension functions 64
testing 71
using CDML Converter to create 42
using in web site or program 45
using layout information in 57
using Site Assistant to create 40
XML-stylesheet processing instruction 36
XSLT, described 37
-styletype query parameter 87
summary of steps for
XML data access 22
XSLT publishing 39
switching layouts for an XML response 35

testing
web sites 71
XML output 71
text encoding
-encoding query parameter 52, 82
encoding settings 52
for XSLT requests 52
generated XML data 26
initial default settings 52
request and output pages default 52
URLs 25, 59
using string manipulation extension functions 64
time extension functions, using 66
time format strings 68
-token query parameter 54, 88
tools for XSLT, described 11, 40, 42
transforming data with stylesheets 37
troubleshooting
Custom Web Publishing web sites 71
XML document access 36
XSLT stylesheets 46

U
Unicode characters 33
URL syntax for
container objects in XML solutions 24
container objects in XSLT solutions 48
XML requests 23
XSLT stylesheets 48
URL text encoding 25
url_decode() extension function 64, 65
url_encode() extension function 64
US-ASCII encoding 52
user names
access to XML documents 57
Basic Authentication for web users 17, 56
defining for Custom Web Publishing 17
User-Agent header, checking 54
UTF-8 (Unicode Transformation 8 Bit)
coding setting 52
format 25, 33

V
values, checking for in checkboxes 65
vCards, reformatting XML data into 21
-view query command 81

W
web browsers
receiving output 9
role in XML requests 22
role in XSLT-CWP requests 38
Web folder, copying container field objects 19
Web Publishing Core
illustrated 22, 38
internal access logs 73
Web Publishing Engine
Administration Console 16, 18, 22, 39
application log 73
benefits of 10
described 9
Development mode 95
generated error codes 89
generating pages from XSLT stylesheet 38
generating XML data 22
generating XML documents 23
Production mode 96
Web Security Databases, not supported 15
web server
log files 72
MIME type support 18
role in XML requests 22
role in XSLT-CWP requests 38
web sites
creating with Web Publishing Engine 10
FileMaker support pages 10
monitoring 72
testing 71
web users
accessing protected databases 17, 56, 57
requirements for accessing Custom Web Publishing solutions 12
using container field data 19
web_server_module_log.txt log file 73
wpc_access_log.txt file 73

X

XML
described 21
document type definitions (DTDs) 26, 27, 30
encoding using UTF-8 format 26, 33
filtering data 21
FMXMLLAYOUT grammar 31
FMXMLRESULT grammar 30
fmresultset grammar 27
generating XML data from request 22
grammars, described 25
namespaces for 26
order of request processing 35
parsers 23, 33
query strings 33, 75
requesting data 23
summary of steps for accessing XML data 22
troubleshooting access to XML documents 36
URL text encoding 25
using client-side stylesheets 36
XML 1.0 specification 21
XML-stylesheet processing instruction 36
XPath statements 54
<xsl:stylesheet> element 50, 54, 55, 71, 100
<xsl:message> element 70
<xsl:output> element 53
<xsl:param> element 54

<xsl:param name="authenticated-xml-base-uri"/>
parameter 56
<xsl:param name="client-ip"/> parameter 56
<xsl:param name="client-password"/> parameter 56
<xsl:param name="client-user-name"/> parameter 56
<xsl:param name="request-query"/> parameter 55
<xsl:param name="xml-base-uri"/> parameter 56
<xsl:template> element 56, 71, 72, 100, 103
<xsl:variable> element 57

XSLT
CDML Converter, using 42
checkboxes, checking for values in 65
comparing strings via Perl regular expressions 65
content buffering, using 58
cookie extension functions 63
date and time format strings 68
date, time, and day extension functions 66
described 37
developing stylesheets 47
e-mail messages, sending 61
enabling in database 17
error status of extension functions, checking 70
examples of stylesheets 37
Extension Function Reference 48
extension functions for FileMaker 54
FileMaker-specific XSLT parameters 54
generating pages from XSLT stylesheet 38
-grammar parameter 50
header functions, using 62
layout information, using 57
namespaces for 50
query strings for 49
Query Strings Reference 77
server-side stylesheets 37, 47
Site Assistant, using 40
string manipulation extension functions 64
summary of steps for publishing 39
troubleshooting stylesheets 46
using stylesheets in web site or program 45
XSLT 1.0 specification 37
XSLT-CWP requests 38
xslt-template-files folder 39, 45, 57
<?xslt-cwp-buffer buffer-content="true"?> processing instruction 58
<?xslt-cwp-query?> processing instruction 47, 51
xslt-template-files folder 39, 45, 57