Lasso 2.5

The Ultimate FileMaker® Pro Web Development Tool

blueworld

bringing business to the internet™

Blue World Communications, Inc.
10900 NE 8th Street, Suite 1525
Bellevue, Washington 98004 U.S.A.

Tel: (425) 646-0288 Fax: (425) 646-0236

blueworld@blueworld.com

http://www.blueworld.com

Contents

Chapter 1: New Features in Lasso 2.5

There have been many internal changes to Lasso offering improved performance, enhanced functionality, new interoperability between tags, and greater standardization of tag operations.

Plug-in Architecture: Lasso 2.5 introduces a plug-in architecture which allows one to enhance Lasso's functionality and compatibility simply by placing module files into the "Lasso Modules" folder.

Math Tags: The Math Tags module offers six mathematical tags which perform arithmetic calculations.

String Tags: The String Tags module provides several tags which perform powerful text parsing and manipulation.

FileMaker 4.0 Pro Tag Support: The FMP4 Tags module handles compatibility with Claris FileMaker Pro 4.0 Web Companion. Web Companion tags (CDML) can be used with Lasso (LDML) to seamlessly upgrade solutions.

Field-Level Logical Operators: Using the -opbegin and -opend command tags, field-level logical operators are now available, including the NOT operator.

Multihoming: Lasso 2.5 Server supports multihoming, allowing multiple domain names to operate smoothly on a single machine.

Lasso Startup Items: Lasso 2.5 adds support for a startup items folder which launches items prior to starting Lasso.

Tags Within Tags: Any substitution tag can now be used as a value or parameter to any other substitution tag. For instance, one can output the data from a field whose name is specified by another field.

Relative Paths: Path names can now be relative. Paths in format and include files without a beginning slash ("/") are assumed to be relative.

Response Field: A database field may be specified to format the response by simply referencing the field name in the response command. A new "ResponseField" security option is added to the Lasso Security database.

Usernames and Passwords: Usernames and passwords are submitted using a browser authentication dialog or by using the new -lassousername and -lassopassword command tags.

Over 50 New Tags: LDML (Lasso Dynamic Markup Language) is now considerably larger, with over 200 tags in total. Some examples include:

- **[post_inline: …]** — Executes inlines after the currently processed page is finished and returned to the user.
- **[total_records]** — Displays the total number of records in the current database.
- **[response_file_path]** — Displays the name of the current response file.
- **[field_name]** — Displays the the names of fields.
- **[var: …]** — Allows for variable declarations within a page for conditional value checking.
- **[while: …]** — Allows for conditional looping. As long as the conditional returns true, the loop continues.
- **[lasso_process: …]** — Allows Lasso to process any tags located within the parameter strings for the tag.

Standardization: Additional standards are now adhered to.

- **UTF Encoding** — Database names are now UTF decoded when layout names are requested (Java only).
- **PIXO Support** — Lasso now allows other Web server plug-ins to send data to Lasso for processing.
- **MIME Handling** — When a directory path is placed in the Lasso Server "Processed Items" window, only files with the MIME type of "text/*" (where * is any MIME subtype) are processed.
- **Keep Alive** — Lasso Server now supports the Keep Alive convention for increased performance.
- **HTTP Headers** — The HEAD method is now understood.
- **CLF Standard** — Added support for Common Log Format (CLF) to Lasso Server.
- **Realm Security** — Lasso Plug-in and CGI can now use WebStar realms to restrict file access.

Specifying FileMaker Scripts: FileMaker Pro scripts may now be referenced before or after calling Lasso, and may operate in the foreground or background.

Error Handling: Several improvements have been made to Lasso's error handling. Value list information is now available in error pages. Errors are logged to a "LassoErrors.log" file. A default error page can be set for Lasso Server.

File Structure Flexibility: The Lasso Plug-in now works when the Web server does not live at the root of the document tree.

Next and Previous Links: The links created by the [next], [prev], and [referrer] container tags can now be displayed on the response file using the new [next_url], [prev_url], and [referrer_url] substitution tags.

Multiple Parameter Retrieval: Specific tag parameters are now retrieved. For example, the phrase [form_param: paramName, 2] will return the second parameter.

Value Counting: The number of values for a specified form parameter can now be retrieved with the count keyword.

Sending Apple Events: The [event_result], [event_resultcode], and [event_errorstring] tags can refer to data from the preceding Apple Event tag instead of the last Apple Event tag on the page.

Syntax: Several changes in preferred syntax have been made.

- Action and command tags are now preceded with a dash ("-") rather than surrounded with square brackets.
- When using "list_value" in a conditional statement to see if the value is "checked," one must now use the following form: [if: list_value, checked != ""], which will evaluate to true if the list value is checked, false otherwise.
- The beginning and ending of parameter tags can now be enclosed between parentheses. This is particularly helpful in clarifying the use of tags within tags.

Chapter 2: Introduction

General Requirements

The following items are required for Lasso 2.5:

- **Power Macintosh (or compatible) computer** (68020-68040 Macintosh computers are not supported).
- 24 **MB RAM** available for the basic application set: Mac OS, the Web server, FileMaker Pro and Lasso application. (16 MB is acceptable for Mac OS 7.1 – 7.6.1).
- **Mac OS 7.1 or greater.**
- **FileMaker Pro 3.0v4 or greater** (client edition, not to be confused with FileMaker Pro server edition). All Roman-based language editions of FileMaker Pro are supported.
- **Mac OS Web server** which follows the StarNine Technologies' specifications as follows:

  For Lasso CGI: WebSTAR CGI communications standard (which includes virtually all Mac OS Web servers).

  For Lasso Plug-in: WebSTAR API (plug-in specification).

  For Lasso Server: No additional Web server is required.
- **clip2gif 0.7.2** (a freeware application which must be obtained separately) if on-the-fly conversion of graphics stored in a FileMaker Pro database to GIF format is required.

System Extensions

- **AppleScript** ™ (standard with System 7.1 or greater).
- **ObjectSupportLib 1.2** (for Mac OS 7.1 to 7.6.1; not necessary with Mac OS 8 or above as this library is now built-in).
- **Finder Scripting Extension** (for Mac OS 7.1 to 7.5.5; not necessary with Mac OS 7.6 or above as this extension is built-in).
- **QuickTime**™ and **QuickTime**™ **PowerPlug** (version 2.5 or greater) if on-the-fly conversion of graphics stored in a FileMaker Pro database to JPEG format is required.

- **Open Transport 1.1.1** or greater (version 1.3 is required for Lasso Server's multihoming feature). Note: With Lasso CGI or Plug-in, MacTCP is acceptable.

What's Included on the CD

Lasso Editions Included

Lasso 2.5 is available in three editions: CGI, Plug-in and Server.

1. **Lasso Plug-in** — Lasso Plug-in implements version 1.0 of the WebSTAR API plugin specification for extending the functionality of Web servers, and therefore requires a Web server which uses this API version 1.0.
2. **Lasso.acgi —** Lasso CGI (Common Gateway Interface) is an "asynchronous CGI," thus the suffix ".acgi" is used. Lasso.acgi application runs as a separate application from your Mac OS Web server application, and requires your Web server follow the WebSTAR CGI communications standard.
3. **Lasso Server** — Lasso Server is a high-performance HTTP server integrated with all the features of Lasso available in the CGI/ Plug-in. Beyond the basics, Lasso Server offers high-end features such as multihoming, multiple format logging, support for Keep Alive, startup items, and more. Lasso Server is the best option to use when the Web server's primary task is to serve FileMaker Pro databases via Lasso.

Other Lasso Components

- **Modules** — Lasso 2.5 provides a plug-in architecture, allowing separate module files to be integrated into any of the above Lasso editions.
- **LassoCommonCode** — The "LassoCommonCode" file contains core code common to all editions of Lasso. This required file works with the Lasso application as well as modules to provide all of Lasso's functionality.
- **Lasso Security Databases** — These databases contain security configuration information.

Other Lasso Files Included

Also included are additional files and information to get you started with Lasso 2.5:

- **FM Link** — FM Link is an authoring tool for rapidly building Lasso format files.
- **Employees Example** — An employees example database and corresponding format files are provided to demonstrate key Lasso features.
- **Database Template** — A template database is provided.
- **Lasso Tag Converter** — This BBEdit plug-in allows one to easily upgrade format files or update files from FileMaker Pro 4.0.

Setup Q & A

Q: Should I use Lasso CGI, Plug-in, or Server? The decision depends upon many factors, some of which are described below.

Lasso Server: Lasso Server is a high-performance HTTP server especially optimized for serving FileMaker Pro databases via Lasso. The integrated architecture makes this the overall best performing option.

CGI or Plugin: The main differences between the Lasso Plug-in and CGI editions concern the amount of data that can be transferred.

The Lasso Plug-in provides all features of the Lasso CGI, except that the Lasso Plug-in does not support the -timeout parameter (the Lasso Plug-in uses your Web server's timeout value).

Q: Can Lasso be used with FileMaker Pro 4.0 acting as the Web server? FileMaker Pro 4.0 cannot be used as a server with any other CGI/plug-in. However, Lasso can integrate with its database functionality in the same manner as with FileMaker Pro 3.0.

Q: How many hits can a PowerPC Mac OS Web server handle at one time? For example, is there a limit to how many ports it can have open at a time? The number of hits is dependent upon the number of available concurrent ports. With Open Transport about 150 ports can be used simultaneously. With MacTCP (not compatible with Lasso Server), about 50 ports can be open at one time. However, some of these ports may be dedicated to other applications.

Q: How much Web traffic can a Mac OS Web server with Lasso handle? Many Lasso integrated Web sites are known to handle hundreds of thousands of hits per day. For details on how to optimize your Web server for Lasso databases, refer to OPTIMIZING PERFORMANCE in APPENDIX A: TIPS AND TECHNIQUES.

Q: How many databases can Lasso serve at one time? Lasso is restricted by the number of databases that can be opened by FileMaker Pro, which is currently 50.

Q: How many users can connect to the same database at the same time? Database "connections" on the Web are entirely different than database connections in the traditional sense. Unlike with FileMaker Pro Server, a Web-enabled database can handle as many users as resources allow. Connections are established with each Web request, not via login. Consequently, the number of simultaneous users is only limited by the number of simultaneous requests. Since Lasso is multi-threaded, it can handle numerous simultaneous requests. FileMaker Pro is not multi-threaded; it must handle requests one at a time. Lasso queues requests as it waits for a response from FileMaker Pro.

Usage Rights

Your license permits a single copy of Lasso to be installed and used on a single CPU. You may install Lasso on as many CPUs as you have purchased Lasso serial numbers. You may install and use the FM Link application on as many machines as you desire.

The Lasso serial number is network protected. This means that only one copy of Lasso with the same serial number can run on the same AppleTalk network.

Chapter 3: Installation and Setup

Lasso CGI

Installation

- Copy the "Lasso.acgi" and "LassoCommonCode" files found in the CGI folder in the Lasso package to any location within your Mac OS Web server folder. Note: Lasso.acgi and LassoCommonCode *must* be kept together at the same level in your directory structure.
- Copy the "Lasso Modules" folder found in the CGI folder in the Lasso package to the same location as the "Lasso.acgi" and "LassoCommonCode" files.
- Copy the "Lasso Startup Items" folder found in the CGI folder in the Lasso package to the same level as "Lasso.acgi." This folder contains the Lasso Security databases and must be installed for Lasso Security to operate. If you do not copy this folder, Lasso will create "Lasso Startup Items" as an empty folder when it is first launched, without the databases required for security. Note: To properly protect items placed within the "Lasso Startup Items" folder, you must set up a security realm for this folder. To set up the realm, consult your Web server manual.
- If you intend to use remote administration of Lasso Security, copy the "Security" folder found in the CGI folder in the Lasso package to the same location as Lasso.acgi. All of the HTML files found within the "Security" folder must remain within this folder, unaltered.
- Copy the "Databases" folder found in the CGI folder in the Lasso package to any location on the drive the Web server resides on. For security reasons, do not place databases within your Web serving folder. Be sure to open all Web-enabled databases prior to launching Lasso (unless they reside in the "Lasso Startup Items" folder).
- Copy the "Employees" folder found in the CGI folder in the Lasso package to the same location as Lasso.acgi.

- Copy the "FM Link" and "User Guide" folders to your drive; no specific location is required. The contents of each must remain intact.

Registration

Launch the "Lasso.acgi" application and enter your serial number into the dialog box. Click "OK" to register the software. Quit and relaunch "Lasso.acgi" to store the serial number within the application.

Lasso Plug-in

Installation

- Copy the "Lasso Plug-in" file found in the Plug-in folder in the Lasso package to the Plug-ins folder of your Mac OS Web server folder. If the Lasso Plug-in is installed properly it will be automatically loaded when you launch your Mac OS Web server (and noted in the Web server's log window).
- Copy the "LassoCommonCode" file and "Lasso Modules" folder found in the Plug-in folder in the Lasso package to the same location as your Web server application.
- Copy the "Lasso Startup Items" folder found in the Plug-in folder in the Lasso package to the same location as your Web server. If you do not copy this folder, Lasso will create "Lasso Startup Items" as an empty folder when it is first launched, without the databases required for security.
- If you intend to use remote administration of Lasso Security, copy the "Security" folder found in the Plug-in folder in the Lasso package to the same location as your Web server. All of the HTML files found within the "Security" folder must remain within this folder, unaltered.
- If you intend to use remote administration of Lasso Security, copy the "Security" folder found in the server folder to the root level. All of the HTML files found within the "Security" folder must remain within this folder, unaltered.
- Copy the "Databases" folder found in the Plug-in folder in the Lasso package to any location on the drive the Web server resides on. For security reasons, do not place databases within your Web serving folder. Be sure to open all Web-enabled databases prior to launching Lasso (unless they reside in the "Lasso Startup Items" folder).

- Copy the "Employees" folder found in the Plug-in folder to the root level of your Web server.
- Copy the "FM Link" and "User Guide" folders to your drive; no specific location is required. The contents of each must remain intact.

Registration

When Lasso receives the first action through your Mac OS Web server you will be prompted to enter your Lasso serial number into the dialog box. Click "OK" to register the software. Quit and relaunch your Mac OS Web server to store the serial number.

Lasso Server

Installation

- Copy the "Lasso Server" and "LassoCommonCode" files found in the Server folder in the Lasso package to a new folder designated for Web serving. This folder may reside anywhere on your drive, and will be considered the "root level" of your Web serving environment. *Note:* Lasso Server and LassoCommonCode *must* be kept together at the same level in your directory structure.
- Copy the "Lasso Modules" folder found in the Server folder in the Lasso package to the root level of your new Web serving folder.
- Copy the "Lasso Startup Items" folder found in the Server folder in the Lasso package to the root level of your Web serving folder (the same level as Lasso Server). This folder contains the Lasso Security databases and must be installed for Lasso Security to operate. If you do not copy this folder, Lasso will create "Lasso Startup Items" as an empty folder when it is first launched, without the databases required for security. Note: Lasso Server denies requests to download files contained within the "Lasso Startup Items" folder. Databases and other sensitive files may be securely stored there.
- If you intend to use remote administration of Lasso Security, copy the "Security" folder found in the Server folder to the root level. All of the HTML files found within the "Security" folder must remain within this folder, unaltered.
- Copy the "Databases" folder found in the Server folder in the Lasso package to any location on the drive the Web server

resides on. For security reasons, do not place databases within your Web serving folder. Be sure to open all Web-enabled databases prior to launching Lasso (unless they reside in the "Lasso Startup Items" folder).

- Copy the "Employees" folder found in the Server folder to the root level of your Web server.
- Copy the "FM Link" and "User Guide" folders to your drive; no location is required. The contents of each must remain intact.

Registration

Launch the Lasso Server application and enter your serial number into the dialog box. Click "OK" to register the software. Quit and relaunch Lasso Server to store the serial number within the application.

Lasso Modules

- *Lasso Tags Module:* This required module contains the basic core set of Lasso tags.
- *Math Tags Module:* This module offers six mathematical tags which perform arithmetic calculations.
- *String Tags Module:* This module offers several tags which perform powerful text parsing and manipulation.
- *Log Tag Module:* This module logs Lasso activity to a targeted file or database.
- *Database Info Tags Module:* This optional module provides information about open databases. Note: Due to possible security concerns, this module is optional.
- *Encoding Tags Module:* This optional module performs raw, url, smart, standard encoding and url decoding.
- *FMP4 Module:* This optional module provides compatibility with Claris FileMaker 4.0 Web Companion and CDML tags.
- *Non-Relative Response Tag and Include Tag Modules:* These optional modules provide backward compatibility with -response and [include] tags as they were handled prior to Lasso 2.5.

- *Apple Event Tags Module:* This optional module provides tags that allow Apple Events to be sent from Lasso. Note: Due to possible security concerns, this module is optional.
- *Old If Tag Module:* This optional module provides backward compatibility with the way the [if: …] tag was handled prior to Lasso 2.5.

Configuring Modules

Similar to how Mac OS extensions operate, the order in which Lasso modules load is important. The functionality within one module may add to or replace like functionality in other modules that load before it. Lasso must be relaunched after installing or removing modules.

Lasso Startup Items

The "Lasso Startup Items" folder can be used to launch specific applications, open databases, or to process format files when Lasso initially launches. It is located in the same folder as the Lasso application (if it does not exist, this folder will be created when Lasso is first launched). When Lasso starts up, it will open every item in the folder. If the item is a FileMaker Pro database, Lasso will wait until the database is accessible before opening the following items or initializing security. This allows one to place the Lasso Security databases in this folder and have them completely opened before Lasso proceeds to initialize security. If the item is a text file with the suffix ".lasso" (or ".las"), Lasso will process the page as a format file. Thus, one can execute database activity immediately after the server has started.

Upgrading from Other Versions of Lasso

Any Lasso solution that was created with a previous version of Lasso is easily upgraded to Lasso 2.5 without extensive modifications. For the most part, the principles and construction of format files are the same in all versions of Lasso. However, since there have been syntax changes during the development of Lasso, a few adjustments will be necessary. For adjustments that apply across numerous Lasso format files, we recommend using a tool such as BBEdit which allows one to rapidly search and replace text strings among multiple text files. For your convenience, we have included a demo of BBEdit 4.5 in the Lasso package, although until it is purchased you will not be able to save changes to documents.

Lasso Tag Converter

To make upgrading easier, included in the Lasso package is a BBEdit Plug-in called Lasso Tag Converter.

Lasso Tag Converter upgrades Lasso format files for all versions prior to 2.5. In addition, Lasso Tag Converter converts FileMaker Pro 4.0 CDML format files to Lasso 2.5 LDML.

- The Lasso Tag Converter plug-in requires BBEdit 4.0 or higher. A demo version of BBEdit is included in the Lasso package, or it can be obtained from:

  http://www.barebones.com/

- To install the Lasso Tag Converter, place it in your "BBEdit Plug-in" (or "BBEdit Extensions") folder, and restart BBEdit. In the "Tools" (or "Extensions") menu, you will now see the Lasso Tag Converter.
- Open an old Lasso format file, or a FileMaker Pro 4 format file, and choose the "Lasso Tag Converter" item from BBEdit's "Tools" (or "Extensions") menu.
- The Lasso Tag Converter will quickly convert all tags to their Lasso 2.5 equivalent.
- The Lasso Tag Converter will alert you of any tag it is unable to process, allowing you to simply click on "warnings" to have the questioned instance highlighted for easy editing.
- New ("Untitled") documents can be processed by the Lasso Tag Converter, but warnings will not be triggered. It is always preferable to convert saved documents. Changes made to your documents will **not** be saved automatically, and BBEdit's "Undo" feature is fully supported.
- The Lasso Tag Converter will always convert entire documents, not just the current selection.
- There are over 200 string searches, permutations and replacements accomplished by the Lasso Tag Converter. Processing documents with a large number of lines can take a bit of time. If enough memory is available, a progress dialog will appear during long operations. Document processing can be stopped by pressing command and period (.) together or clicking the progress dialog "Cancel" button. The Lasso Tag Converter will then finish the current search/replace, then stop.

  Note: Make sure that BBEdit's windows are not set to "soft wrap text" or the warning browser window will highlight incorrect lines.

Converting from CGI to Plug-In

To migrate from the CGI to the Plug-in edition, a simple change to your format files is needed.

The primary difference between format files used with Lasso.acgi and Lasso Plug-in is how the form action is specified. Simply perform a global find on "Lasso.acgi" and replace with "action.lasso." Change from:

```
<form action="Lasso.acgi" method="post">
```

To:

```
<form action="action.lasso" method="post">
```

For pre-Lasso form submissions (depending upon how your files were previously set up to operate with Lasso CGI), you may need to change the path to access the Lasso Plug-in in the form action.

Upgrading Security Databases to Lasso 2.5

The "Lasso Security" database and remote security format files have been updated for Lasso 2.5. As new security functionality has been introduced, you must use the new databases for security to operate properly. Import the data from your previous copy of the "Lasso Security" database into the new "Lasso Security" database, using matching fields. Also, replace the previous "Security" folder with the new "Security" folder. The "Security" folder contains the HTML format files used for remote administration of the "Lasso Security" databases.

FileMaker Pro Setup

FileMaker Pro databases do not need to be modified for use with Lasso. However, to improve performance (sometimes dramatically), create basic FileMaker Pro layouts which correspond with the fields used in your Lasso format files (especially when working with more sophisticated or larger databases). Target your Lasso format files to link to specific FileMaker Pro layouts which contain only the necessary fields.

The following rules apply:

All Lasso-enabled FileMaker databases:

- *must* be opened by FileMaker Pro before a Lasso form action is called.

- *must* be opened on the same CPU as the Mac OS Web server in the *client* version of FileMaker Pro (as opposed to the FileMaker Pro *server* application). The FileMaker Pro Server version does not contain the full suite of Apple Event support found in the FileMaker client. For further information, consult FILEMAKER PRO SERVER in APPENDIX A: TIPS AND TECHNIQUES.
- *must* be in browse mode (to allow Apple Events to be sent, for better performance, and for FM Link to operate; but also for general performance).
- *should* be opened in single-user mode, but can be used as "guest" to a "multi-user" database, as the "host" of a multi-user database, or as a guest to a database hosted by FileMaker Pro Server.
- *should* be opened with the master password if passwords have been established, specifically such that all necessary fields, layouts, scripts, and functions (browsing, printing, exporting, creating records, etc.) are available. For more details, see error -10004 and -10005 in the "Error Codes" section of the FileMaker Pro User Guide.

Chapter 4: Lasso Methodology

General Overview

Lasso operates by interpreting special codes, referred to as "Lasso tags" or "LDML" (Lasso Dynamic Markup Language), placed in simple text files referred to as Lasso "format files." Combined with HTML, LDML provides the instructions Lasso uses to communicate with FileMaker Pro and a Web server. Communication occurs based upon actions, commands and the types of tags specified.

Tags are used in different ways depending upon whether the format file is accessed before Lasso is called (pre-Lasso) or after Lasso is called (post-Lasso). As data moves from a visitor's Web browser to a Web server, through Lasso, to a database, and then back, the data passes through many applications which use different protocols for handling data. Consequently, data is translated ("encoded" and "decoded") as it passes through the process.

Format files may contain other format files ("includes") or contain embedded operations ("inlines"). Data values can be manipulated irrespective of a database (with Math and String tags), and sending email and other operations can also occur with no database interaction. FIGURE 1: HOW LASSO EXCHANGES DATA shows various ways Lasso exchanges data.

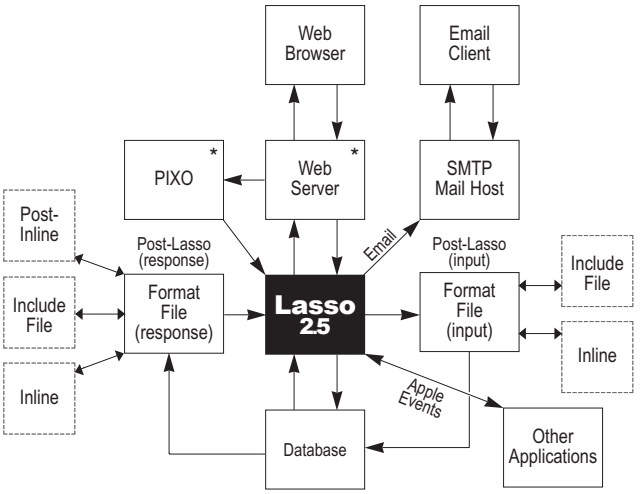
Format Files

Format files are templates Lasso uses to format the data it sends and retrieves to and from FileMaker Pro databases. Format files also contain all the instructions for handling non-database operations.

The basic format file types are as follows:

- **Add —** Adds new records to a database.
- **Add Reply —** Acknowledges a successful record addition.
- **Add Error —** Sends an error page due to data entry errors when attempting to add a record.
- **Update —** Updates pre-existing records.

Figure 1: How Lasso Exchanges Data



* Unnecessary if Lasso 2.5 Server is used.

- **Delete Reply —** Acknowledges a successful deletion of a record.
- **Search —** Searches records.
- **Search Results —** Displays the results of a search.
- **Detail —** Displays the details of a single record from the database. Delete and Duplicate actions may appear here.
- **Update Reply —** Acknowledges a successful update.
- **Duplicate Reply —** Acknowledges a successful record duplication.
- **No Results Error —** Returns an error when a search provides no results.

Referencing Format Files Stored in a Database Field

Rather than accessing format files stored on disk, Lasso can reference format file data stored within a FileMaker Pro database field. The syntax is as follows:

```
-response=field:fieldname
```

You must first assign "ResponseField" permissions in Lasso Security before any targeted field can serve this function.

Editing Format Files

As some HTML editors do not support extended HTML (as in LDML), use extreme caution when editing format files in anything but a text editor.

Lasso Tags

There are five types of Lasso tags: Action, Command, Substitution, Container and Sub-Container.

Action Tags

(-add, -search, -show, etc.)

Action tags direct Lasso to perform a specific action when a format file is submitted. Generally, a single action occurs with each form submission. However, using the [inline: …] tag, several actions can occur in a single form submission.

Action tags are specified within an HTML "submit" button, as names within an HTML hidden input or within a form action statement. For example, the submit button for an "Add" action appears as follows:

```
<input type="Submit" name="-add" value="Add Record">
```

For all actions, the name of the submit button must be an action tag. The value can be any text you want to appear on the button. Images may also be used as submit buttons, as in the following example:

```
<input type="Image" src="searchgraphic.gif" name="-search">
```

If no action is specified, Lasso defaults to the -nothing action. Lasso will still process the page without conducting any database activity.

Command Tags

(-database, -response, -emailto, etc.)

Command tags instruct Lasso on how to define the specified action. Command tags are specified as hidden input names or may appear within a form action statement

Substitution Tags

([field: …], [server_date: …], [include: …], etc.)

Substitution tags are placeholders for where data is inserted. The data may originate from a database, from the HTTP header of the current request, a separate file, or represent a value from a previous submission. Substitution tags can take other tags to their value or parameters.

Container Tags

([if: …]…[/if], [record: …]…[/record], [inline: …]…[/inline], etc.)

Container tags function as substitution tags, but are different because they require an opening and closing tag pair. Some Lasso container tags allow HTML and other Lasso substitution tags to reside within their tag pair.

Sub-Container Tags

([else], [list_value], [repeat_value], etc.)

Sub-container tags operate within select container tags. Like substitution tags, sub-containers can take other tags for their value or parameters.

Basic Rules

In terms of syntax, LDML recognizes some basic rules:

- Action and command tags begin with a dash ("-"; e.g., -add or -database).
- Substitution, container, and sub-container tags are surrounded by square brackets (e.g., [record] or [field: …]).
- Container and sub-container tags are similar to HTML tags in that they have opening and closing tag components (e.g., [record]…[/record]).
- Sub-containers, as indicated by their name, are used only in conjunction with container tags, between container tags' opening and closing components.
- Some tags — particularly substitution tags — can be used within other tags. In this case, the embedded tag is not surrounded by brackets or preceded by a dash.
- LDML is not case-sensitive; i.e., Lasso will recognize "-database" or "-Database" or "-dataBase.

Components of a Lasso Tag

Lasso tags are constructed and operate in a manner analogous to HTML form elements. HTML form elements operate by specifying a "name," "value" and "type." The corresponding items in LDML are "tag name," "value" and "parameters." The following represents the basic structure applicable to the majority of Lasso substitution, container and sub-container tags.

```
[tag: "value", parameter]
```

The tag structure is explained as follows:

- **name** — A tag name is required.
- **value**— The value can be a literal text string (must be surrounded by quotes) or another tag (not surrounded by quotes).
- **parameter**— There are four types of parameters.
    1. **Keywords:** Some keywords work with many tags as parameters (e.g., "count"); others work with a single tag (e.g., "jpeg" works only with [image: …]).
    2. **Encoding Keywords:** The "raw," "url, "break" and "smart" encoding keywords work with all substitution tags to apply character encoding.
    3. **Other Tags:** Substitution and sub-container tags can function as parameters within other tags. Quotes should not be used.
    4. **Literal String Value:** Literal strings must be surrounded by quotes.

Tags Within Tags

It is possible to use Lasso substitution tags within any other substitution or sub-container tag. The "embedded" tag becomes a value or parameter of the "embedding" tag. For instance, one can output the data from a field whose name is specified by another field:

```
[field: (field: "InsertField")]
```

To clarify which values or parameters belong to which tag, parentheses should be used.

Parentheses are required if an embedded tag uses more than one parameter. For example:

```
[inline: database="main", layout="web" name=(field: name,row), search]
```

Or:

```
[if: (field: "name",raw)=="john"]
```

Using Name=Value Pairs

The following rules apply to tags that use "name=value" pairs ([if: …], [inline: …], [set_var: …], [post_inline: …], [set_cookie: …], and all of the math and string tags):

- The name=value pairs and the Lasso action are separated by commas (rather than ampersands as with embedded URLs).
- Input fields are indicated by the name of the field in quotes.
- Substitution fields using the "field" tag to substitute the current value of a field must use the tag name "field" followed by a colon (i.e., field:category).
- Do not use any square brackets within other square brackets. Similarly, do not use hyphens before action or command tags within square brackets.

Pre-Lasso vs Post-Lasso

"Pre" means **before** a Lasso action is processed, and "post" refers to **after** a page is processed . The distinction is actually rather simple, and important to keep in mind when determining how format files and appropriate tags are referenced.

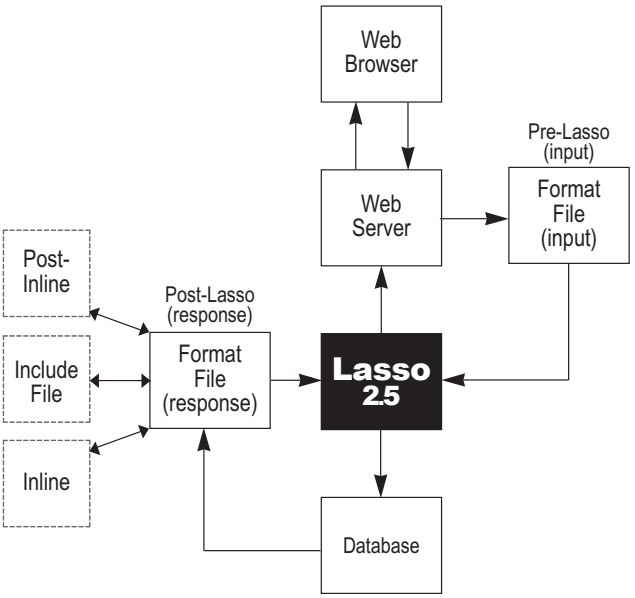
Pre-Lasso: Files Not Served Via Lasso

With the pre-Lasso method, HTML pages are referenced through a direct hyperlink, or URL. A standard HTML page, which has not first been parsed by Lasso, does not contain any dynamic data (from FileMaker Pro or any other source). Field values must be hard-coded into the HTML page. Pre-Lasso files (basic search or add forms) may reside anywhere on the Internet.

Post-Lasso: Files Served Via Lasso

With the post-Lasso method Lasso builds forms and HTML pages "on-the-fly" using elements found in a FileMaker Pro database. The post-Lasso form is parsed by Lasso, and then returned to the server for display. On a post-Lasso form, field values may be auto-populated with values from a database. The post-Lasso method applies to all Lasso format files that define how data is retrieved or displayed as a result of a Lasso action. Post-Lasso format files must be located on a volume mounted on the Mac OS Web server, and accessible by AppleTalk.

Figure 2: Pre-Lasso Form Input



An Example of Pre-Lasso and Post-Lasso Methods

In the provided "Employees" example, the "add2.html" and "search2.html" files represent a post-Lasso method of serving forms via Lasso, while the "add.html" and "search.html" files represent a pre-Lasso method of serving forms.

- Pre-Lasso — The files "search.html" and "add.html" can be directly called up to view a complete form. Here is how one would retrieve the "search.html" form:

  ```
  <a href="http://www.your_server.com/Employees/search.html>Search</a>
  ```

- Post-Lasso — Lasso generates "add2.html" and "search2.html" forms using data from the specified FileMaker Pro database using the -show command. Here is how you would retrieve the "search2.html" form:

  ```
  <a href="http://www.your_server.com/Lasso.acgi?-database=Employees&
  -layout=Detail&-response=Example/search2.html&-show">Search</a>
  ```

Specifying Paths

This section discusses methods for specifying paths to Lasso actions for the various editions of Lasso, and discusses how to reference Lasso format files. Note: With Lasso 2.5, Lasso now recognizes relative path names.

Pre-Lasso

Lasso.acgi

The best way to reference Lasso.acgi in a pre-Lasso file is to place an alias of the Lasso.acgi application in your project folder and rename it "Lasso.acgi."

To indicate the path to an example folder, use the following:

```
<form action="/example/Lasso.acgi" method="post"> or
<a href="/example/Lasso.acgi?-database=…"
```

This allows for relative references to be used throughout for all the format files stored within the "Example" folder, thus simplifying the creation of path names within these format files.

It is recommended that you locate the Lasso.acgi in the same folder as the Web server application (to simplify the process of defining path names); however, if it is buried in folders, a path from the HTML file to Lasso must be indicated. For example:

```
<form action="/cgi-bin/Lasso.acgi" method="post"> or
<a href="/cgi-bin/Lasso.acgi?-database=…">
```

Lasso Server and Plug-in

The best way to reference the Lasso Server or Plug-in on a pre-Lasso file is to specify the path to the project folder as follows:

```
<form action="/example/action.lasso" method="post"> or
<a href="/example/action.lasso?-database=…">
```

Although the server requires only the "action.lasso" portion to direct the action to Lasso, the path that is included makes it possible to use relative references to all format files from this point on.

Alternately, Lasso can be referenced from the root level as follows:

```
<form action="/action.lasso" method="post"> or
<a href="/action.lasso?-database=…">
```

Note: A slash is used to indicate that the path begins at the root level of the Web serving folder.

Absolute

Lasso can be specified with an absolute location, for example:

```
<form action="http://www.domain.com/web server/Lasso.acgi" method="post">
```

Post-Lasso

All post-Lasso calls are either "Lasso.acgi" or "action.lasso" since Lasso is considered to be the source of the document.

For the Lasso Server and Plug-in:

```
<form action="action.lasso" method="post"> or
<a href="action.lasso?-database=…">
```

For Lasso.acgi:

```
<form action="Lasso.acgi" method="post"> or
<a href="Lasso.acgi?-database=…">
```

Referencing Format Files

The following guidelines can be used for specifying paths to files referenced by Lasso. The -response tag requires a valid path, as it specifies the name of the format file to be used to format the successful response of a Lasso action.

Use the following guidelines when specifying a path:

- The name of files and directories are not case-sensitive.
- Relative path names are supported.
- Format files are usually located only within the Web serving folder. However, Lasso can recognize format files stored outside of the Web serving folder if the filename ends with ".lasso" or ".las."
- A format file on another server cannot be referenced unless the file resides on a volume mounted on the primary server via AppleTalk and the filename ends with ".lasso" or ".las."
- The path to a format file can refer to an alias of the format file.
- An absolute path can be used if it specifies all directories that lead to the referenced file.

Note: In versions of Lasso prior to 2.5, post-Lasso format files were specified from the location of the Lasso application rather than relative to the previously called file.

Calling Lasso

With a Form

To submit a form, you generally specify actions and commands within the form action statement. For example:

```
<form action="action.lasso?-database=Employees&-layout=summary&
-search"> ...rest of the HTML form </form>
```

In a form action statement, the action should always be the last tag specified. As some older browsers may not support actions specified within form statements, you can also specify actions and commands within hidden HTML input fields as follows:

```
<form>
<input type="hidden" name="-database" value="Employees">
<input type="hidden" name="-layout" value="Summary">
rest of the HTML form…
<input type="Submit" name="-add" value="Add Record">
</form>
</body>
</html>
```

With a Link

Lasso actions can be embedded in a URL that is executed when a hyperlink is selected. For example:

```
<a href="http://domain.com/webserver/Lasso.acgi?-database=
database_name&…etc">
```

An embedded URL strings together Lasso tags and other "name=value" pairs to specify the values and parameters for the action. When Lasso performs a search, it automatically creates embedded URLs for the [next] and [prev] links, as well as links to detailed records.

When constructing embedded URLs, please keep the following in mind:

- Tags which modify other tags (such as the -operator tag) must always appear before the tags they affect.
- All data encoded in a hyperlink is encoded as "name=value" pairs, separated by an equals sign ("="):

  ```
  name=value
  ```

- "Name=value" pairs are separated with ampersands:

```
name1=value1&name2=value2&name3-value3&nameN=valueN...
```

- The name of a reference database is always required:

  ```
  -database=DatabaseName
  ```

- The name of a reference layout is not required, but recommended:

  ```
  -layout=LayoutName
  ```

  (Lasso cannot search summary fields, portals, calculation fields, auto-entered fields, etc., if a layout name is not specified.)

For example, to search a sample "Gurus" database using the layout named "Detail" for all records where the field "Last Name" contains "Cusick," return five records at a time starting with record 11 of the result set, sort the records by "Last Name," and have the results formatted based on file "Detail Format.html," the URL would appear as follows:

```
<a href="action.lasso?-database=Gurus&-layout=Detail&-response=
Detail+Format.html& -op=cn&Last+Name=Cusick&-maxRecords=5&
-skiprecords =10&-sortfield=Last+Name&-search"> Search </a>
```

Note: Blank spaces are encoded with a plus sign ("+"). See APPENDIX B: CHARACTER SETS AND TRANSLATION for details.

Chapter 5: Getting Started

This chapter describes the FM Link tool and provides a tutorial designed to help beginners get up and running quickly.

FM Link

Figure 3: FM Link

Specially created for use with Lasso, FM Link assists with the creation of Lasso format files. Complementing your text editor or HTML authoring program, FM Link provides a palette of LDML tags and dynamically retrieves data from open FileMaker Pro databases.

FM Link queries any open FileMaker Pro database and generates a list of related databases, layouts, fields and scripts.

The "Lasso Tags" tab presents a comprehensive listing of Lasso tags. Detailed information for a specific tag is obtained simply by double-clicking on the tag's name.

Drag Menu

The following options are available under the "Drag" menu:

As Text/As HTML

There are two options for the method used when dragging items from the FM Link palette. Text can be dragged: "As Text," "As HTML," or both (if both are checked). If using a standard text editor, leave both selected.

These selections were introduced to allow FM Link to operate with certain HTML authoring applications. The problem arises when dragging the HTML content of the various menu items into the GUI design mode of some HTML authoring tools. Symbols used to create HTML are encoded and appear as literal text instead of as HTML. To correct this, deselect the "Text" item so that content is dragged as HTML only.

Include Field Labels

When "Include Field Labels" is selected, the name of the field will be included, along with a colon, followed by the HTML used for the field entry and/or the Lasso tags used to display that field. This drag option is only enabled when the "Fields" tab is selected.

By deselecting this option, the field name will not be included. The two modes are toggled using the menu or the command-L key combination.

Add Carriage Returns

When "Add Carriage Returns" is selected, a carriage return is automatically inserted after the item is dragged from the FM Link palette. This drag option applies to all items in all tabs. Thus if two items are dragged in succession, each will be placed on their own line. This is particularly useful when dragging fields using the "Post-Lasso (Display)" mode.

By deselecting this option, a carriage return will not be inserted. This is useful when dragging menu items into pre-existing HTML containers or tags, for example, between the quotes of a value equation. The option to add a carriage return can be toggled using the menu or the command-R key combination.

Drag-and-Drop Options

Dragging the text of any menu item from the FM Link window will drag the text or HTML associated with that selection, as well as a carriage return. The content of what is dragged is described below.

Option-drag (select the option key while dragging) items to drag just the text shown in the FM Link window. Any database, layout, field, list value, script, or Lasso tag item is inserted without the corresponding tag syntax or carriage return.

How to Use FM Link

1. Open your FileMaker Pro Database

Open the FileMaker Pro database to be used with Lasso. The database must be open and in "browse" mode. It should remain open and in browse mode when using FM Link.

2. Select a Database

- Drag the database name from the list into a text editor; the Lasso -database command tag will be included with the correct database name inserted.

- Option-drag to drag a specific database name.
- Click the "Update" button to update the list of databases after opening/closing databases.
- Double-click the database name to proceed to the "Layouts" tab.

3. Select a Layout

- Drag one of the layouts listed into a text editor and the Lasso -layout command tag will be included with the correct database name inserted.
- Option-drag to drag a specific layout name.
- Click the "Update" button to update the list of layouts after creating/deleting layouts in FileMaker Pro.
- Double-click the layout name to proceed to the "Fields" tab.

4. Selecting Fields/Values

- In the "Fields" tab, choose an HTML field type from the "HTML Type" pop-up menu.
- Next, select the "mode" this page will be used for. If the file is accessed as a standard HTML file, the "Pre-Lasso" tab should be used; if the format file is processed by Lasso and field values are gathered from a FileMaker database, then either of the "post-Lasso" modes should be used. The display type will simply display the field value on the response page; the update selection will place this field value into HTML input tags for use on an update form.
- If a field does not have a value list associated with it in the specified layout, then the only options are "Text," and "Text Area." If a field has a value list, all selection list types are available for that field (pop-up, scrolling list, radio button, and check box).
- Select a field name from the fields list and drag to the text editor. All value list items are selected by default. To option-drag the name of just a single value, first deselect all values by command-clicking on the white space in "Values" list and then option-drag the value name.
- Option-drag to drag a specific field name or value list item name.
- Click the "Update" button to update the list of fields and values after editing fields and values in FileMaker Pro.

5. Selecting Scripts

In the "Scripts" tab, choose the "mode" a script will be used for. The choices are:

- **Pre —** The script will execute before any other operations are completed.
- **Pre-Sort —** The script will execute after all other operations are completed, but before any sorts take place.
- **Post —** The script will execute after all operations are completed.
- **Pre.Back, Pre-Sort.Back, and Post.Back —** The script executes either before other actions, prior to a sort or after all actions, and operates in the background.

6. Lasso Tags

In the "Lasso Tags" tab, choose a tag name and drag it to your text editor. The indicated Lasso tag with appropriate HTML code is inserted.

- Option-drag to drag just the Lasso tag name.
- Double-click on a tag name to open the Lasso tags help window. (HTML code may also be dragged from the help window.)

7. Templates

The following templates are available in FM Link:

- Add Record
- Add Record Reply
- Add Record Error
- Search Records
- Search Results
- No Search Results
- Detail Record
- Update/Delete/ Duplicate Record
- Delete Reply
- Update Reply
- Duplicate Reply
- Email Template
- Email Message Template

Tutorial: Web-Enable a Basic Database

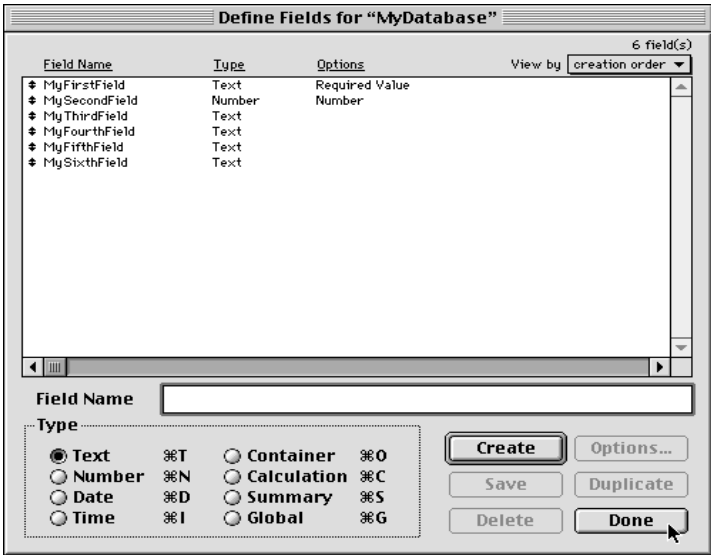
This tutorial provides step-by-step instructions for Web-enabling a basic database, using the "MyDatabase.fp3" database supplied in the "Databases" folder, FM Link, the format file templates listed above, and a text editor. After completing this tutorial you will understand how to add, search, update, delete and duplicate FileMaker Pro Web database records and how to handle errors.

The next series of sections are designed to be read linearly; detailed explanations are not repeated for each step. For more details on each of the tags mentioned, refer to the provided "Lasso 2.5 Reference" database. In this manual, HTML authoring packages are grouped with other text editors and singularly referred to as "text editor."

Sample Database

A sample database, named "MyDatabase.fp3," is provided in the "Databases" folder included with Lasso. "MyDatabase.fp3" is a simple file with one layout, named "MyLayout," and six fields (one number field and five textual ones), as shown in FIGURE 4: SET UP DATABASE.

Figure 4: Set Up Database



Setup

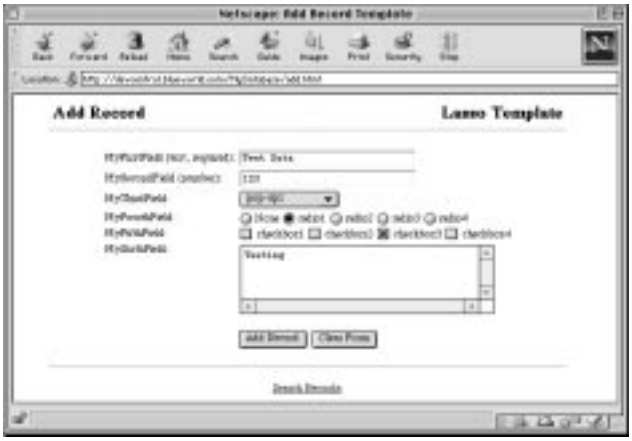
1. Create a folder called "MyDatabase" at the root level of your Web server.
2. Launch FileMaker Pro and open the database "MyDatabase.fp3."
3. Launch Lasso, a text editor, a Web browser, and FM Link.

   Note: All examples below are constructed for Lasso Plug-in or Lasso Server editions. They are easily modified for the Lasso CGI as discussed elsewhere.

Add Records

1. Using your text editor, create a new document named "add.html."
2. Go to the template section of the FM Link palette (select the "Lasso Tags" tab and scroll to the bottom). Select "Add Record Template," and drag it to the empty "add.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Figure 5: Add Record



Add Record Template

```
<html>
<head>
<title>Add Record Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
      <td align=left valign=middle><b><font size="+2">Add
Record</font></b></td>
      <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
   </tr>
</table>
<hr width=550>
 <form action="action.lasso?-add" method="post">
```

```
<input type="hidden" name="-database" value="MyDatabase.fp3">
<input type="hidden" name="-layout" value="MyLayout">
<input type="hidden" name="-response" value="addreply.html">
<input type="hidden" name="-addError" value="adderror.html">
<table cellspacing="0" cellpadding="1">
   <tr>
      <td valign=middle>MyFirstField (text, required):</td>
      <td align=left valign=middle>
         <input type="text" size=30 name="MyFirstField"></td>
   </tr>
   <tr>
      <td valign=middle>MySecondField (number):</td>
      <td align=left valign=middle>
         <input type="text" size=30 name="MySecondField"></td>
   </tr>
   <tr>
      <td valign=middle>MyThirdField:</td>
      <td align=left valign=middle>
         <select name="MyThirdField" size=1>
         <option value="" selected>
         <option value="pop-up1">pop-up1
         <option value="pop-up2">pop-up2
         <option value="pop-up3">pop-up3
         <option value="pop-up4">pop-up4
         </select></td>
   </tr>
   <tr>
      <td valign=middle>MyFourthField:</td>
      <td align=left valign=middle>
         <input type="radio" name="MyFourthField" value="" checked>None
         <input type="radio" name="MyFourthField" value="radio1">radio1
         <input type="radio" name="MyFourthField" value="radio2">radio2
         <input type="radio" name="MyFourthField" value="radio3">radio3
         <input type="radio" name="MyFourthField" value="radio4">radio4</td>
   </tr>
   <tr>
      <td valign=middle>MyFifthField:</td>
      <td align=left valign=middle>
         <input type="checkbox" name="MyFifthField"
value="checkbox1">checkbox1
         <input type="checkbox" name="MyFifthField"
value="checkbox2">checkbox2
         <input type="checkbox" name="MyFifthField"
value="checkbox3">checkbox3
         <input type="checkbox" name="MyFifthField"
```

```
value="checkbox4">checkbox4</td>
   </tr>
   <tr>
      <td valign=top>MySixthField:</td>
      <td align=left valign=top>
         <textarea name="MySixthField" rows=4 cols=40
wrap=soft></textarea></td>
   </tr>
</table>
<p>
<input type="Submit" name="-add" value="Add Record">
<input type="Reset" value="Clear Form"><br>
</form>
<p>
<hr width=550><br>
<a href="search.html">Search Records</a>
</center>
</body>
</html>
```

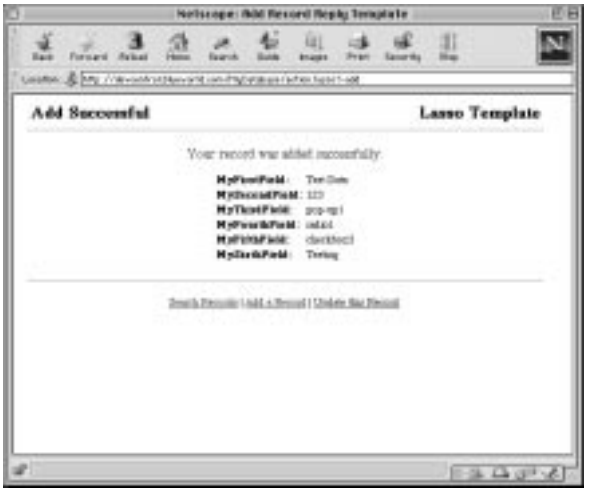
Let's look at the Lasso action and tags used in detail:

- The syntax "action.lasso" designates Lasso as the recipient of the form, and the syntax "?-add" tells Lasso what action to perform when the form is submitted.
- The command tag -database instructs Lasso to use the database "MyDatabase.fp3." This is accomplished using an HTML input statement to pair the -database tag with a value, that is, the name of the database.
- The command tag -layout instructs Lasso to use the layout "MyLayout." This is accomplished using an HTML input statement to pair the -layout tag with a value, that is, the name of the layout.
- The command tag -response instructs Lasso which format file to use to process or display the result of a form submission (in this case, "addreply.html").
- The command tag -adderror instructs Lasso which format file to use to display a custom error page if a user did not fill out the form correctly.
- The action tag -add (used in this form as the name of the submit button) instructs Lasso to add a record to the database. While this was already specified in your form statement, some browsers require the action to appear as the last item on a page.

Add Record Reply

Create a new document in your text editor, drag from FM Link the "Add Record Reply Template," saving it as "addreply.html" into the "MyDatabase" folder.

Figure 6: Add Record Reply



Add Record Reply Template

```
<html>
<head>
<title>Add Record Reply Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
    <tr>
        <td align=left valign=middle><b><font size="+2">Add Successful</font></b></td>
        <td align=right valign=middle><b><font size="+2">Lasso Template</font></b></td>
    </tr>
</table>
<hr width=550>
<blockquote><font size="+1" color="blue">Your record was added
successfully.</font><br>
</blockquote>
<table cellspacing="0" cellpadding="1">
    <tr>
        <td align=left valign=middle><b>MyFirstField:</b></td>
```

```
        <td valign=middle>[field:"MyFirstField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MySecondField:</b></td>
        <td valign=middle>[field:"MySecondField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MyThirdField:</b></td>
        <td valign=middle>[field:"MyThirdField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MyFourthField:</b></td>
        <td valign=middle>[field:"MyFourthField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MyFifthField:</b></td>
        <td valign=middle>[field:"MyFifthField"]</td>
    </tr>
    <tr>
        <td align=left valign=top><b>MySixthField:</b></td>
        <td valign=middle>[field:"MySixthField",break]</td>
    </tr>
</table>
<p>
<hr width=550><br>
<a href="search.html">Search Records</a> |
<a href="add.html">Add a Record</a> |
<a href="[detail_link:layout="mylayout",response="update.html"]">Update this Record</a>
</center>
</body>
</html>
```

You'll notice that only two new Lasso tags are used in this source code — the [field: …] tag and the [detail_link: …] tag.

The [field: …] tag simply references the specified field and the [detail_link: …] inserts the current record ID into the URL calling the update format file when selected.

Now that you've created both an "Add" and "Add Reply" format file, you are ready to try it out. Making sure that Lasso and your Web server are running, and that the files you have created are not open in your text editor, enter the following URL into your Web browser:

http://www.YourServerName.com/MyDatabase/add.html

Note: If you have any difficulty successfully launching your Web server, read the "Testing Without a Network Connection" tip found in APPENDIX A: TIPS AND TECHNIQUES.

Input and select values for **all** the fields and then submit the form by clicking the "Add Record" button.

Add Record Error

Notice that the first two fields in "MyDatabase.fp3" have certain FileMaker Pro options selected. "MyFirstField," a standard text field, has the validation option "required" checked. "MySecondField," a numeric field, has the validation option "of type number" selected. Lasso recognizes these settings and generates errors if users do not comply.

To test this, re-submit the "Add Record" form leaving the "MyFirstField" field blank.

Figure 7: Add Record Error

To create the format file that generated this page, create a new document named "adderror.html." In the template section of FM Link, locate "Add Error Template" and drag it to the empty "adderror.html" document in the text editor. Save this file to the "MyDatabase" folder.

Add Record Error Template

```
<html>
<head>
<title>Add Record Error Template</title>
</head>
```

```
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
      <td align=left valign=middle><b><font size="+2">Add Record
Error</font></b></td>
      <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
   </tr>
</table>
<hr width=550>

<blockquote><font size="+1" color="red">The record was <b>not</b>
added.</font><br>
Select your browser's &quot;Back&quot; key
and check that your submission is complete<br>and that all
data entered is of the correct format<br>(i.e., numbers are numbers,
dates are in mm/dd/yy format).</blockquote>
<p>
<hr width=550><br>
</center>
</body>
</html>
```

Search Records

1. Create a new document named "search.html."
2. Go to the template section of the FM Link palette and select "Search Template," dragging it to the empty "search.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

You now have a pre-Lasso format file. It is pre-Lasso because it is directly called up from a URL without interacting with Lasso.

Search Records Template

```
<html>
<head>
<title>Search Records Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
```

Figure 8: Search Records


```
      <td align=left valign=middle><b><font size="+2">Search
Records</font></b></td>
      <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
   </tr>
</table>
<hr width=550>
<form action="action.lasso?-search" method="post">
<input type="hidden" name="-database" value="MyDatabase.fp3">
<input type="hidden" name="-layout" value="MyLayout">
<input type="hidden" name="-response" value="searchresults.html">
<input type="hidden" name="-noresults" value="noresults.html">
<table cellspacing="0" cellpadding="1">
   <tr>
      <td valign=middle>MyFirstField (text):</td>
      <td align=left valign=middle><select name="-op">
         <option value="bw" selected>begins with
         <option value="eq">equals
         <option value="cn" >contains
         <option value="ew">ends with
         </select></td>
      <td align=left valign=middle><input type="text" size=30
name="MyFirstField"></td>
   </tr>
   <tr>
      <td valign=middle>MySecondField (number):</td>
```

```
      <td align=left valign=middle><select name="-op">
         <option value="eq" selected>equals
         <option value="gt"> &gt;
         <option value="gte"> &gt; or =
         <option value="lt"> &lt;
         <option value="lte"> &lt; or =
         </select></td>
      <td align=left valign=middle><input type="text" size=30
name="MySecondField"></td>
   </tr>
   <tr>
      <td valign=middle>MyThirdField:</td>
      <td valign=middle>(select from pop-up list)</td>
      <td align=left valign=middle><select name="MyThirdField" size=1>
         <option value="" selected>
         <option>pop-up1
         <option>pop-up2
         <option>pop-up3
         <option>pop-up4
         </select></td>
   </tr>
   <tr>
      <td valign=middle>MyFourthField:</td>
      <td colspan="2" align=left valign=middle><input type="radio"
name="MyFifthField" value="" checked>None
         <input type="radio" name="MyFourthField" value="radio1">radio1
         <input type="radio" name="MyFourthField" value="radio2">radio2
         <input type="radio" name="MyFourthField" value="radio3">radio3
         <input type="radio" name="MyFourthField" value="radio4">radio4</td>
   </tr>
   <tr>
      <td valign=middle>MyFifthField:</td>
      <td colspan="2" align=left valign=middle><input type="checkbox"
name="MySixthField" value="checkbox1">checkbox1
         <input type="checkbox" name="MyFifthField"
value="checkbox2">checkbox2
         <input type="checkbox" name="MyFifthField"
value="checkbox3">checkbox3
         <input type="checkbox" name="MyFifthField"
value="checkbox4">checkbox4</td>
   </tr>
   <tr>
      <td valign=middle>MySixthField:</td>
      <td colspan="2" align=left valign=middle><font size="-
1">(contains)</font>
```

```
        <input type="hidden" name="-operator" value="cn"><input type="text"
size=30 name="MySixthField"></td>
    </tr>
    <tr>
        <td colspan="3" align=left valign=middle><hr></td>
    </tr>
    <tr>
        <td valign=middle> </td>
        <td align=center valign=middle>
            <p><input type="Submit" name="-search" value="Start Search"></td>
        <td align=center valign=middle>
            <input type="Submit" name="-findall" value="Find All"> 
            <input type="Submit" name="-random" value="Random"> 
            <input type="Reset" value="Clear Form"></td>
    </tr>
    <tr>
        <td colspan="3" align=left valign=middle><hr>
            <br>Select Optional Search Parameters</td>
    </tr>
    <tr>
        <td valign=middle>Sort By:</td>
        <td align=left valign=middle><select name="-sortfield" size=1>
            <option selected>Unsorted
            <option value="MyFirstField">MyFirstField
            <option value="MySecondField">MySecondField
            <option value="MyThirdField">MyThirdField
            <option value="MyFourthField">MyFourthField
            <option value="MyFifthField">MyFifthField
            </select></td>
        <td align=left valign=middle><select name="-sortorder">
            <option selected>ascending
            <option>descending
            <option>custom
            </select></td>
    </tr>
    <tr>
        <td valign=middle>Max Records:</td>
        <td align=left valign=middle><select name="-maxRecords" size=1>
            <option>5
            <option selected>10
            <option>20
            <option>30
            <option>50
            <option>all
            </select></td>
```

```
        <td valign=middle> </td>
    </tr>
    <tr>
        <td valign=middle>Logical Operator:</td>
        <td align=left valign=middle><input type=radio name="-logicalop"
value="and" checked>AND
            <input type=radio name="-logicalop" value="or">OR</td>
    </tr>
</table>
</form>
<p>
<hr width=550><br>
<a href="add.html">Add a Record</a>
</center>
</body>
</html>
```

Notice that the majority of the code is HTML. Note the following points:

- The search operators Lasso uses correspond to FileMaker Pro's search operators (eq, gt, gte, lt, lte, cn, bw, and ew).
- The syntax "action.lasso" designates Lasso as the recipient of the form, and the syntax "?-search" instructs Lasso to perform the search action.
- The -noresults command tag specifies which format file to use for the response if a user's search yields no results.
- The -operator command tag determines how Lasso compares the submitted field values.
- The -maxrecords command tag specifies the number of hits to show on the response page.
- The -sortfield command tag displays the name of the field used to sort the current found set of records.
- The -findall command tag finds all records in a FileMaker Pro database.
- The -random command tag finds a random record in a database.
- The -logicalop command tag selects a global operator ("and" or "or") to search by.

Search Results

Create a search results (sometimes called a "hitlist") template to display the results of the search.

1. Create a new document named "searchresults.html."
2. Go to the template section of the FM Link palette and select "Search Results Template," dragging it to the empty "searchresults.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Figure 9: Search Results

Search Results Template

```
<html>
<head>
<title>Search Results Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
      <td align=left valign=middle><b><font size="+2">Search
Results</font></b></td>
      <td align=right><b><font size="+2">Lasso Template</font></b></td>
   </tr>
</table>
<hr width=550>
```

```
<p>
Displaying records <b>[begNum]</b> thru <b>[endNum]</b>
(<b>[nShown]</b> records displayed).<br>
(<b>[nFound]</b> records found out of <b>[total_records]</b> in database)
<p>
<b>[prev] |Previous| [/prev] [next] |Next| [/next]</b><br>
<p>
Select the link in MyFirstField to view more details about that record.
<p><br>
<table cellpadding=2>
   <th>MyFirstField</th>
   <th>MySecondField</th>
   <th>MyThirdField</th>
   <th>MyFourthField</th>
   <th>MyFifthField</th>
   <th>MySixthField</th>
   <th>[record]</th>
   <tr>
      <td align="left">
      <a
href="[detail_link:layout="MyLayout",response="Detail.html"]">[field:MyFirstFiel
d]</a>
      </td>
      <td align="center">[field:"MySecondField"]</td>
      <td align="left">[field:"MyThirdField"]</td>
      <td align="left">[field:"MyFourthField"]</td>
      <td align="left">[field:"MyFifthField"]</td>
      <td align="left">[field:"MySixthField"]</td>
      <td>[/record]</td>
   </tr>
</table>
<p>
<hr width=550><br>
<a href="add.html">Add a Record</a>
</center>
</body>
</html>
```

This file introduces eight new Lasso tags ([begnum], [endnum], [total_records], [nfound], [nshown], [prev], [next], and [record]).

- The [begnum] tag displays the numeric range in a found set of records.
- The [endnum] tag displays the ending number in a range of a found set of records.

- The [total_records] tag inserts the total number of records in the database used in the previous Lasso action.
- The [nfound] tag displays the total number of records found.
- The [nshown] tag displays the total number of records in a subset of returned records.
- The [prev] tag displays the previous group in the found set of records.
- The [next] tag displays the next group in the found set of records.
- The [record] tag displays elements for every record returned from a database.

No Search Results

1. Create a new document named "nosearchresults.html."
2. Go to the template section of the FM Link palette and select "No Results Template," dragging it to the empty "nosearchresults.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Figure 10: No Results

No Results Template

```
<html>
<head>
<title>No Results Template</title>
</head>
```

```
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
      <td align=left valign=middle><b><font size="+2">No Search
Results</font></b></td>
      <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
   </tr>
</table>
<hr width=550>
<font size="+1" color="red">
<blockquote><b>No</b> records were found which match your search
criteria.</font>
<br>Select your browser's &amp;quot;Back&amp;quot; key and try your search
again.</font>
</blockquote>
<p>
<hr width=550>
</center></body>
</html>
```

Record Detail

If a search is successful, the user can click on a link to display details for any of the found records.

1. Using your text editor, create a new document named "detail.html."
2. Go to the template section of the FM Link palette and select "Record Detail Template," dragging it to the empty "detail.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Record Detail Template

```
<html>
<head>
<title>Record Detail Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
      <td align=left valign=middle><b><font size="+2">Record
Detail</font></b></td>
```

Figure 11: Record Detail


```
        <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
    </tr>
</table>
<hr width=550>
<table cellspacing="0" cellpadding="1">
    <tr>
        <td align=left valign=middle><b>MyFirstField:</b></td>
        <td valign=middle>[field:"MyFirstField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MySecondField:</b></td>
        <td valign=middle>[field:"MySecondField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MyThirdField:</b></td>
        <td valign=middle>[field:"MyThirdField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MyFourthField:</b></td>
        <td valign=middle>[field:"MyFourthField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MyFifthField:</b></td>
        <td valign=middle>[field:"MyFifthField"]</td>
    </tr>
    <tr>
        <td align=left valign=top><b>MySixthField:</b></td>
```

```
        <td valign=middle>[field:"MySixthField",break]</td>
    </tr>
</table>
<p>
<hr width=550><br>
<a href="search.html">Search Records</a> |
<a href="add.html">Add a Record</a> |
<a href="[detail_link:layout="mylayout",response="update.html"]">Update this
Record</a>
</center>
</body>
</html>
```

Update/Delete/Duplicate Record

Create an update format file which also contains buttons to duplicate or delete a record.

1. Using your text editor, create a new document named "update.html."
2. Go to the template section of the FM Link palette and select "Update Record Template," dragging it to the empty "update.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Update Record Template

```
<html>
<head>
<title>Update Record Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
    <tr>
        <td align=left valign=middle><b><font size="+2">Update
Record</font></b></td>
        <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
    </tr>
</table>
<hr width=550>
<form action="action.lasso?-update" method="post">
<input type="hidden" name="-database" value="MyDatabase.fp3">
<input type="hidden" name="-layout" value="MyLayout">
```

Figure 12: Update Record


```
<input type="hidden" name="-response" value="updatereply.html">
<input type="hidden" name="-recid" value="[recid_value]">
<input type="hidden" name="-duplicatereply" value="duplicatereply.html">
<input type="hidden" name="-deletereply" value="deletereply.html">
<table cellspacing="0" cellpadding="1">
   <tr>
      <td>MyFirstField (text):</td><td>
         <input type=text size=30 name="MyFirstField" value="[field:
"MyFirstField"]">
      </td>
   </tr>
   <tr>
      <td valign=middle>MySecondField (number):</td>
      <td align=left valign=middle>
         <input type=text size=30 name="MySecondField" value="[field:
"mysecondfield"]">
         </td>
   </tr>
   <tr>
      <td valign=middle>MyThirdField:</td>
      <td align=left valign=middle>
         <select name="MyThirdField"><option value="">[option:
"MyThirdField"]</select>
         </td>
   </tr>
   <tr>
      <td valign=middle>MyFourthField:</td>
```

```
        <td align=left valign=middle>
        <input type="radio" name="MyFourthField" value="">None
           [value_list: "MyFourthField"]
              <input type="radio" name="MyFourthField" value="[list_value]"
[checked]>[list_value]
           [/value_list]
</td>
   </tr>
   <tr>
      <td valign=middle>MyFifthField:</td>
      <td align=left valign=middle>
         [value_list: "MyFifthField"]
            <input type="checkbox" name="MyFifthField" value="[list_value]"
[checked]>[list_value]
         [/value_list]
      <input type="hidden" name="MyFifthField" value="">
</td>
   </tr>
   <tr>
      <td valign=top>MySixthField:</td>
      <td align=left valign=top>
         <textarea name="MySixthField" rows=4 cols=30
wrap=soft>[field:"MySixthField"]</textarea>
</td>
   </tr>
</table>
<p>
<input type="Submit" name="-update" value="Update">
<input type="Submit" name="-duplicate" value="Duplicate">
<input type="Submit" name="-delete" value="Delete">
<input type="Reset" value="Reset Form">
</form>
<p>
<hr width=550><br>
<a href="search.html">Search Records</a> |
<a href="add.html">Add a Record</a>
</center>
</body>
</html>
```

Again, the majority of the code is HTML. Ten Lasso tags are used: (-update, -recid, -duplicate, -duplicatereply, -delete, -deletereply, [option: …], [value_list: …], [list_value], and [checked]).

Keep in mind the following points when reviewing the code:

- The -recid command tag instructs Lasso which record to update. It is paired with the [recid_value] tag, which is substituted with the record ID for a specific record in the database when this file is processed.
- The -duplicate action tag instructs Lasso to make a copy of the current record in the database.
- The -duplicatereply command tag specifies which format file to use after a successful deletion.
- The -delete action tag instructs Lasso to delete the current record from the database.
- The -deletereply command tag specifies which format file to use after a successful deletion.
- The [option: …] substitution tag is used within selection lists to indicate where data is to be submitted.
- The [value_list: …] container, along with the [list_value] sub-container, displays values from a FileMaker Pro value list.
- The [checked] sub-container tag places the HTML "checked" tag into the format file so that any items currently selected for this record display as checked items on the form.

Update Reply

To indicate that a database record update was successful, create an update reply template:

1. Using your text editor, create a new document named "updatereply.html."
2. Go to the template section of the FM Link palette and select "Update Reply Template," dragging it to the empty "updatereply.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Update Reply Template

```
<html>
<head>
<title>Update Reply Template</title>
</head>
<body bgcolor=white>
<center>
```

Figure 13: Update Reply


```
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
      <td align=left valign=middle><b><font size="+2">Update
Successful</font></b></td>
      <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
   </tr>
</table>
<hr width=550>
<blockquote><font size="+1" color="blue">Your record was updated
successfully.</font><br>
</blockquote>
<table cellspacing="0" cellpadding="1">
   <tr>
      <td align=left valign=middle><b>MyFirstField:</b></td>
      <td valign=middle>[field:"MyFirstField"]</td>
   </tr>
   <tr>
      <td align=left valign=middle><b>MySecondField:</b></td>
      <td valign=middle>[field:"MySecondField"]</td>
   </tr>
   <tr>
      <td align=left valign=middle><b>MyThirdField:</b></td>
      <td valign=middle>[field:"MyThirdField"]</td>
   </tr>
   <tr>
      <td align=left valign=middle><b>MyFourthField:</b></td>
```

```
        <td valign=middle>[field:"MyFourthField"]</td>
    </tr>
    <tr>
        <td align=left valign=middle><b>MyFifthField:</b></td>
        <td valign=middle>[field:"MyFifthField"]</td>
    </tr>
    <tr>
        <td align=left valign=top><b>MySixthField:</b></td>
        <td valign=middle>[field:"MySixthField",break]</td>
    </tr>
</table>
<p>
<hr width=550><br>
<a href="search.html">Search Records</a> |
<a href="add.html">Add a Record</a> |
<a href="[detail_link:layout="mylayout",response="update.html"]">Update this
Record</a>
</center>
</body>
</html>
```

Delete Reply

To indicate that a database record deletion was successful, create a delete reply template:

1. Using your text editor, create a new document named "deletereply.html."
2. Go to the template section of the FM Link palette and select "Delete Reply Template," dragging it to the empty "deletereply.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Delete Reply Template

```
<head>
<title>Delete Reply Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
    <tr>
        <td align=left valign=middle><b><font size="+2">Delete
Successful</font></b></td>
```

Figure 14: Delete Reply


```
        <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
    </tr>
</table>
<hr width=550>
<font size="+1" color="blue">
<blockquote>Your record was deleted successfully.</font><br>
</font></blockquote>
<p>
<hr width=550><br>
<a href="search.html">Search Records</a> | <a href="add.html">Add a
Record</a>
</center>
</body>
</html>
```

Duplicate Reply

To indicate that a database record duplication was successful, create a duplicate reply template:

1. Using your text editor, create a new document named "duplicatereply.html."
2. Go to the template section of the FM Link palette and select "Duplicate Reply Template," dragging it to the empty "duplicatereply.html" document in the text editor.
3. Save this file to the "MyDatabase" folder.

Figure 15: Duplicate Reply

Duplicate Reply Template

```
<html>
<head>
<title>Duplicate Reply Template</title>
</head>
<body bgcolor=white>
<center>
<table cellspacing="0" cellpadding="1" width=550>
   <tr>
      <td align=left valign=middle><b><font size="+2">Duplicate
Successful</font></b></td>
      <td align=right valign=middle><b><font size="+2">Lasso
Template</font></b></td>
   </tr>
</table>
<hr width=550>
<font size="+1" color="blue">
<blockquote>Your record was duplicated successfully.</font><br>
</font></blockquote>
<p>
<hr width=550><br>
<a href="search.html">Search Records</a> | <a href="add.html">Add a
Record</a>
</center>
</body>
</html>
```

After you've completed this tutorial and understand the basics of how to create Lasso format files, set up the provided "Employees" example and read the comments provided within the supplied format files.

Chapter 6: Lasso Server

The Lasso Web Server application incorporates all of the Lasso CGI functionality for publishing FileMaker Pro databases on the Web. The Lasso Server includes the following features:

- Ability to serve HTML, JPEG, GIF and multiple MIME file types.
- Support for serving multiple sites, each with its own IP address (multihoming) and configuration options.
- Optimized for maximum efficiency when serving FileMaker Pro databases, outperforming both CGI and Plug-in solutions.
- Enhanced for security.

Lasso Server does not support other CGIs, plug-ins or realm security.

Preparing Format Files

All of the work to prepare your format files is carried out in the same manner as with the Plug-in version of Lasso. Lasso Server's default configuration recognizes the ".lasso" extension. Thus, Lasso knows to direct processing to itself when it receives a request with the extension ".lasso" in the file name portion of the URL. For example:

```
<form action="action.lasso" method="post">
...rest of form
</form>
```

Here is an example of the first part of an embedded URL:

```
<a href="action.lasso?-database=Employees">
```

The location of the Lasso application does not need to be indicated. All response files and file references are considered to be relative to the base URL.

Menu Options

The Lasso Server menus provides control and configuration options.

File Menu

The "File" menu contains options for:

- Displaying or closing the log window.
- Quitting the application.

Server Menu

The "Server" menu contains the following configuration options:

Stop Servers

Servers are started and stopped by toggling between the two choices at the top of the "Server" menu list.

Configuration

This dialog is used to configure a single served site, as well as multihomed sites. The following applies to each served site. Refer to the multihoming section for details on setting up multiple served sites.

- **Active** — If checked, the currently viewed site is active (default is checked).
- **Logging Enabled** — If checked, Lasso will write log details to a file. See the "Log Window" details below (default is checked).
- **Port** — Sets the port used by the server; note that all virtual hosts can use the same port (default is port number 80).
- **Max Connections** — Sets the maximum number of simultaneous connections (default is 15 connections).
- **Timeout** — Sets the time that Lasso will wait for a completed request (default is 60 seconds).
- **TCP Buffer Size** — Sets the size of the TCP packet for sending. (The recommended default for TCP is 8192 bytes).
- **Default File** — Sets the name used as the default file for a specific directory, e.g., "index.html" or "default.html." This file is the one used if a specific file for a path is not specified in a URL.
- **Error File** — Sets the file used when a request is made for a file that does not exist. A colon before the name indicates that the file is to be saved at the root level of the Web serving folder (default is ":error.html").

- **No Access File** — Sets the file used when an unauthorized request is made for a secured file (default is ":noaccess.html").
- **Max. Keep Alive Sessions** — See below (default is 5 sessions).
- **Keep Alive Timeout** — See below (default is 15 seconds).
- **Site Root Folder** — Sets the location of the Web serving folder for a specific served site (default is the folder Lasso currently resides in).

The "default," "error" and "no access" files need to be created and stored within the root folder of a specific site.

Keep Alive

The Keep Alive extension to HTTP, as defined by the HTTP/1.1 draft, allows persistent connections. These long-lived HTTP sessions allow multiple requests to be sent over the same TCP connection, and in some cases have been shown to result in an almost 50% performance increase in latency times for HTML documents containing many images. In order for Keep Alive support to be used, the browser must support it. Many current browsers, including Netscape Navigator 2.0 and higher, and Spyglass Mosaic-based browsers (including Microsoft Internet Explorer) support Keep Alive.

Configuration settings take effect when the "Save Current Site" button is selected. Lasso Server does not need to be restarted. However, when modifying the Server Port or Maximum Connections settings, the Lasso Server application needs to be stopped and then restarted. (Note: This is not to be confused with quitting and relaunching the application, which is not necessary.) To stop the server, select "Stop Servers" from the "Server" menu, then select "Start Servers."

Suffix Mapping

Lasso Server can be set to process and/or serve:

- Files named with certain suffixes.
- Files of a particular MIME type.

Several default suffixes are pre-configured. Lasso will automatically process any file with the following suffixes:

- ".lasso"
- ".cgi"
- ".acgi"

If a file with a suffix that is set to process is requested by a client, the file will first be filtered through Lasso before returning to the client. Lasso will perform any actions specified within the file, including sending email, database activity, etc. Any MIME type that is not set to serve will not be returned to the client.

To add a MIME type to the list:

- Type in the MIME type and file suffix.
- Select the "add" button to add it to the list.

To make changes to an existing MIME type suffix:

- Double-click on the name of the item to be changed.
- Make the desired changes.
- Select the update button.

To remove an item from the list.

- Highlight the item.
- Select the "Remove" button.

When the "Save" button is selected the settings are saved and immediately apply. If the window is closed without saving the changes, the changes are discarded.

If a new file type is set to be processed, the action for an HTML form can be different. For example, the suffix ".html" within an HTML form element can be specified as:

```
<form action="action.html" method="post">
…rest of form
</form>
```

Or:

```
<a href="action.html?-database=Employees">
```

Processed Items

Lasso Server can be configured to process specific files or files in specific folders. All files that are listed in the processed items list are first processed by Lasso before returning to the client. For a folder, any files within that folder, and all its nested folders, are processed by Lasso before returning to the client.

To set the items to be processed, open the "Processed Items" configuration window. Items are added using the buttons at the bottom of the window, or by dragging-and-dropping items to be processed into the window itself.

Log Window

Lasso Server writes current server activity to the log window when the window is open. Logging still occurs when this window is closed, it is just not visible. The log window option allows for a choice between two types of log formats: CLF and Lasso Style. The log file type does not affect the activity of the server.

The standard Lasso Style log appears as follows:

```
11/23/97 6:47:15 PM 209.19.18.252 209.19.18.229/templatefolder/default.html
129 bytes/8 ticks
```

This reports the date, time, IP number of guest, absolute path of file accessed, and the elapsed time for the action to be performed.

The Common Log Format, CLF, is a standard format used for log analysis. It would appear as follows:

```
209.19.18.252 - - [19/Nov/1997:13:31:02 -0800] "GET /Example/green_dot.gif
HTTP/1.0" 304 129

209.19.18.252 - - [19/Nov/1997:13:31:16 -0800] "POST /Example/action.lasso
HTTP/1.0" 200 18054
```

Lasso will also create a log file called "Lasso Server.log" when the "Logging Enabled" checkbox is selected in the "Configuration" dialog. The log file is saved by default to the root level of the Web serving folder. The CLF log style is used as the default logging style for the "Lasso Server.log" file. This cannot be customized or changed to Lasso Style logging.

Multihoming

Lasso Server can provide Web services for multiple virtual hosts on a single computer.

To configure each of the multi-homed sites, open the "Configuration" dialog and select one of the available IP numbers. Custom settings can be assigned to each available IP address with different site root folders to separate the contents of each Web site. Each of the settings described in the "Configuration" section apply to that specific virtual host. Use the "Save All Sites" button to save the settings for all sites at one time.

The multiple IP addresses for your computer are established using Open Transport (1.3 and above). One way to add additional IP addresses is to create a text file called "IP Secondary Addresses." This file needs to list the various IP numbers to be used. Here is an example:

```
; 'ip='  for ip address, 'sm=' subnet mask
; Note: no space in 'ip=17.201.22.200';
; IP address   Subnet Mask
;-----------  -----------
ip=209.19.18.224
ip=209.19.18.225
ip=209.19.18.226
```

(The lines which start with a semi-colon are comment lines.)

Save this file in the Preferences folder within the System Folder, and then restart your computer.

Chapter 7: Security

Lasso security uses a method similar to a Web server's realm-based security to determine whether or not a file can be accessed. Both require that a "username" and "password" be authenticated with a security dialog when a specified item is requested. The difference between the two is that Web server realms protect files or directories, while Lasso security protects databases, fields, records and actions. Once the authenticated username and password are established, they are sent by the browser with every request to the Web server. Lasso checks these values against the settings configured in a set of security databases, and then determines if privileges exist for a specific action.

The Lasso security scheme is configured by assigning usernames, passwords and access privileges in the Lasso security databases. The databases can be modified directly in FileMaker Pro or via Lasso's own remote security administration Web interface.

If installed according to the provided installation instructions, Lasso's default installation is with security enabled. With a standard installation, one must first enable security setting for databases in order to serve them with Lasso.

Lasso Security Databases

Lasso security is controlled by a main database called "Lasso_Security.fp3," and two related databases, called "Lasso_Users.fp3" and "Lasso_Fields.fp3." Lasso reads the security databases when first launched, and initializes, or stores, all security information in RAM.

The security databases are found in the folder named "Lasso Startup Items" within the Lasso distribution package and consequently load automatically when Lasso launches. The databases, however, can be stored anywhere on the computer on which the Lasso application resides.

Two Methods for Configuring Security Settings

Security settings may be configured directly within the provided Lasso Security databases, or from within a Web browser.

1. **Configuring Security Settings Within FileMaker Pro**

   Changing configurations while working directly on a live Web server computer is not recommended, as this will affect the server's performance. In addition, if the Lasso Security databases are modified directly with FileMaker Pro (or via commands sent from other applications), Lasso.acgi and Lasso Server must restart for the changes to take effect. With the Lasso Plug-in, the Web server must be restarted.

2. **Configuring Security Settings Within a Web Browser**

   If a global administrator has been established (see below), all Lasso Security settings can be configured remotely via a Web browser. Changes made this way do not require that Lasso or the Web server restart., and there is no impact to performance on the Web server during configuration. However, unless you are using the Plug-in and an SSL-encrypted Web server, any security information submitted is not encrypted.

Successfully Accessing Security Databases

Upon launch, Lasso attempts to read information from the security databases. If Lasso fails to read the security databases for any reason, Lasso security is disabled. There are four circumstances in which Lasso is unable to read the security databases:

1. FileMaker Pro isn't running when Lasso is launched.
2. The "Lasso_Security.fp3" database isn't open when Lasso is launched.
3. An unexpected error occurs while reading the security databases. Lasso will display an error code in its console window.
4. The Lasso Server, Lasso.acgi, or the Web server (for the Plug-in) application does not have enough memory allocated in order to store the settings stored in the "Lasso_Security.fp3" database. Note that this will only be the case when thousands of users have privileges assigned in these databases.

Essential Configuration Issues

Protecting Lasso Databases

It is essential that the Lasso Security databases are installed correctly such that they cannot be downloaded remotely by anyone on the Web. If the databases are contained within the

"Lasso Startup Items" folder (recommended; default installation), then it is critical that measures are taken to prevent unauthorized access to these databases. This can be accomplished by either establishing a Web server realm or Web server action to deny serving any files with the suffix ".fp3" (the latter approach is recommended).

Note: The Lasso Server is internally configured to deny access to all files containing the ".fp3" suffix.

Protecting Other Databases

All other databases installed on one's Web server should be protected in the same manner that Lasso Security databases are protected. By using a standard file suffix (".fp3") for all databases, and denying direct downloads of these files, it is then quite easy to protect all databases, regardless of where they are installed.

Protecting Lasso Format Files

Similar to methods described above, it is recommended that Lasso format files are protected by either realms or unique file suffixes (for example, ".lasso" or ".fmt").

Setting Up a Global Administrator

The recommended first step for configuring the Lasso Security databases is to set up a global database administrator.. This administrator will have all permissions granted to all databases installed. The global administrator is configured directly within FileMaker Pr.o.

Configuration Keywords

Lasso recognizes two special configuration keywords:

- **All Databases** — The "All Databases" keyword is used as the database name for assigning access privileges (permissions) for all databases served by Lasso. The "Lasso_Security.fp3" database contains a special record called "All Databases." Select users may have certain permissions which apply to all databases they use. Generally, the database administrator has all permissions enabled.
- **All Users** — The "All Users" keyword allows all visitors to a specific database to share a common set of permissions. When a database is assigned "All Users," there is no need to enter a username or password for the assigned actions.

It is possible to provide "All Users" permissions for "All Databases" to provide certain access privileges for all users for all databases (for example, to allow all users the permission to search databases).

Factors Affecting Security

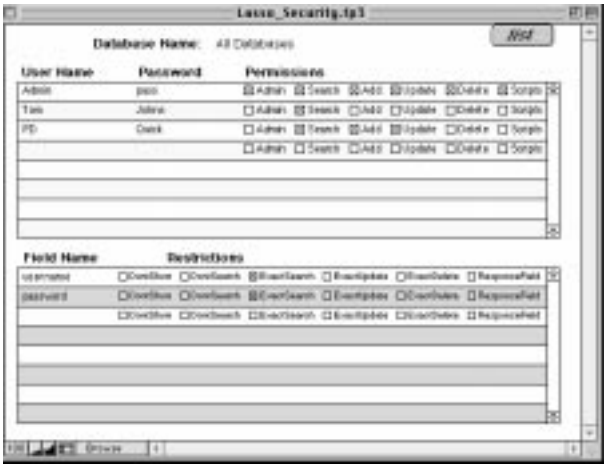
In order for a database to be served via Lasso when security is enabled, one of the following conditions must be true:

1. A record exists in the "Lasso_Security.fp3" database defining usernames, passwords, and permissions for a specific database.
2. A record exists in the "Lasso_Security.fp3" database defining usernames, passwords, and permissions for "All Databases."

In order for a user to access a database served via Lasso when security is enabled, one of the following conditions must be true:

1. The username, password, and permissions are defined for the specified database.
2. The username, password, and permissions are defined for "All Databases."
3. "All Users" have permissions defined for the specified database.
4. "All Users" have permissions defined for "All Databases."

Figure 16: Lasso Security Database — "All Databases" Record



User Authentication

The user authentication dialog is presented when a user initially attempts to access a database protected by Lasso security. The username and password entered into the user authentication dialog must match those entered in the Lasso security databases for the specified database.

Once the user has entered their username and password successfully, the username and password values are sent by the Web browser in every request to a specific Web server. The user will not be presented with the authentication dialog again unless they try to access another database or call an action set with different permissions requiring a new username or password. If the user is prompted for a username and password by the Web server for realm-based security, the values are also included by the browser in requests sent to Lasso. Therefore, if their Lasso username and password are the same as those used by the Web server, the user won't be presented with an authentication dialog again.

Database-Level Security

Access to databases served via Lasso is controlled by assigning usernames, passwords, and permissions in the "Lasso_Security.fp3" database. Database-level security is controlled via the upper portion of the display. The lower portion is used to control field- and record-level security.

Defining Permissions

Perform the following steps from the FileMaker Pro user interface to define usernames, passwords, and permissions for a database served via Lasso:

1. Select the New Record command from the Mode menu. This will create a blank record.
2. Enter the name of the database to be served via Lasso in the "Database Name" field.
3. Enter usernames and passwords in the yellow and white striped portal in the upper portion of the display. Passwords must contain a value.
4. Set the permissions for each user.

There are six possible permissions that can be granted to a user for a database:

- **Admin** — Grants the user permission to use Lasso remote security administration to modify user permissions for the database across the Web. This also exempts the user from any field- or record-level security restrictions defined for the database and automatically grants the user permission to perform the other five actions for the database, even if they are not explicitly granted.
- **Search** — Grants the user permission to search the database, subject to any field- or record-level security restrictions. Includes -show and -findall.
- **Add** — Grants the user permission to add records to the database.
- **Update** — Grants the user permission to update records in the database, subject to any record-level security restrictions.
- **Delete —** Grants the user permission to delete records in the database, subject to any record-level security restrictions.
- **Scripts —** Grants the user permission to execute FileMaker Pro scripts via the -scripts tag, or scripts that appear in forms with other actions via the -doscript tag. If the script is run with another action the user must have permission for that action as well as for scripts. For example, if a script appears in a search form via the -doscript tag and the user doesn't have scripts permission, they won't be able to execute the search.

The -show action is not included on the permission list, but if a user is granted either search or add permission for the specified database, they can also use the -show action to bring up a search or add form. The -findall tag also does not have a selection, but is protected in the same manner as "search."

Field- and Record-Level Security

The pink and white striped portal in the lower portion of the "Lasso_Security.fp3" display is used to control field- and record-level security for a database. These security features are implemented by defining restrictions that limit the use of specified fields in a database. Field- and record-level restrictions may be defined for "All Databases." A "Security Violation" error page is returned to the end user if any of these restrictions are violated and the user does not have sufficient privileges for that database (or "All Databases").

There are three field-level security restrictions that can be defined for any field in any database served via Lasso:

- **DontShow** — Data from the field is not available for display in a Lasso format file except to users with "Admin" privileges for the database. If a field with this restriction appears in a format file, a blank value is returned as if the field was empty.
- **DontSearch —** The field cannot be used in a search form except by users with "Admin" privileges for the database. If a field with this restriction appears in a search form, the field is not used in the search request sent by Lasso to FileMaker Pro.
- **ResponseField —** Enables the specified field to be used as the response for a Lasso action. This method can also be used with any of the following command tags: -response, -duplicatereply, -adderror, -deletereply, -noresults, -reqfieldmissing, and -emailformat.

In addition, there are three record-level security restrictions that can be defined for any field in any database served via Lasso:

- **ExactSearch** — Only records whose fields contain the exact value specified for the search will be returned. When "exactsearch" is assigned to a field, the "equals" operator must be used with that field when it is present in a search action. In addition, if any field is defined as "exactsearch" then the -findall action cannot be used with that database.
- **ExactUpdate —** When updating a record, the value provided by the user for this field must exactly match the value in the database. This has the side effect of preventing updates to fields with this restriction.
- **ExactDelete** — When deleting a record, the value provided by the user for this field must match the value in the database.

If you have any field set with record-level security, the field *must* be included in the form that specifies that action. For example, if two fields are set to be "exactsearch" in the "Lasso_Security.fp3" database, the search form will need to add two text entries for these same fields. A security violation will occur if either of the fields are left blank. The same is true for ExactUpdate and ExactDelete.

Each of the record-level security restrictions can be made on a related field, if the relation is specified with the relationship name followed by a double colon and then the field name. A non-blank value must be specified for the field when searching the database. In addition, the specified value must not contain any FileMaker Pro wildcard or range search characters (*, @, !, =, //, "..", "…", or ellipsis).

Lasso Tags Related to Security

If Lasso Security is initialized, Lasso will check the access privileges of the current user when a request is made to a specific database. The username and password values that are sent by the browser are checked against those stored in the "Lasso Security" database. There are two sets of tags that allow one to submit or display these authentication values for the current request.

Substitution Tags

The Lasso substitution tags [client_username] and [client_password] can be included in a response file to substitute the user authentication values passed by the Web browser. These values can be displayed, saved in a database, or used in a conditional statement to determine the content of the returned format file. For example, one may want to save these values to a activity log using an inline action as follows:

```
[inline: add, database="site_log.fp3", layout="main",
   "PasswordField"=client_password, "User"=client_username,
   "Date"=server_date, "time"=server_time]
[/inline]
```

Note that the [client_username] and [client_password] values are not available if the user authentication dialog was not presented and successfully passed.

Submit Tags

The -lassousername and -lassopassword tags can be used to submit the user authentication values with a Lasso action. In other words, the values used for checking access privileges to a protected database can be set in a Lasso action. The -lassousername tag is used to set the username, while -lassopassword is used to set the password used in the security check. Standard browser authentication will be used if a username and password are required. The value used for these tags can be a hard-coded literal value or a value returned from a database field.

For example, if a "business" database is set to require an exact search for the "name" and "pw" fields, a search can still occur with an embedded URL link using the following syntax:

```
<a href="action.lasso?-search&-database=business&-layout=main&
-response=detail.html&name=[field:"name"]&-lassousername=admin&
-lassopassword=goahead"> Search </a>
```

In another example, the tags are used in an inline action as follows:

```
[inline: database="business", layout="main", "leads"="top",
   lassousername="admin", lassopassword="goahead", search]
      [record]
         Top Leads = [field:"name"]<br>
      [/record]
[/inline]
```

Note that in these examples "admin" and "goahead" are sample values. In this case they would need to be assigned in the "Lasso_Security.fp3" database with, at least, search privileges.

Remote Security Administration

Lasso security can be administered remotely via a Web browser. The HTML format files necessary to accomplish remote administration are located in the folder named "Security." A global administrator is able to make changes to all databases. In addition, administrators for specific databases can be established. The global administrator password must be set directly within FileMaker Pro before the remote security administration can be used.

Lasso recognizes the special username "Admin" (predefined in the "Lasso_Security.fp3" database; see FIGURE 16: LASSO SECURITY DATABASE — "ALL DATABASES" RECORD) for "All Databases." Only the "Admin" user is allowed to add databases in Lasso security via remote administration, and exercise remote administration privileges for "All Databases." Users who have been granted "Admin" permission for a specific database may modify security for the database via remote administration. It is not possible to delete databases from the security setup via Remote Security Administration, but users can be deleted.

Follow these steps to remotely establish security for Lasso databases for either the Lasso Plug-in, Lasso.acgi, or Server version:

1. Make sure that Lasso has been installed correctly and that the "Lasso_Security.fp3" database is open. Check that the "Security" folder is at the root-level of the Web serving folder.
2. No password is predefined for user "Admin." From within FileMaker Pro, open the file named "Lasso_Security.fp3" and locate the password field just to the right of the "Admin" user on the database record for "All Databases" (see FIGURE 16: LASSO SECURITY DATABASE — "ALL DATABASES" RECORD). Insert a password value.

3. Launch either the Lasso.acgi, Lasso Server, or Web server application (for the Lasso Plug-in version).
4. The Remote Security Administration feature is accessed via the "default.html" file in the "Security" folder. Launch the Web browser and go to:

   http://www.yourserver.com/security/default.html

5. From the default page you can either search for a database to administer, or add a new database. To set up a database for the first time, enter the name of the new database name, new username, and new password. Alternately, "All Users" can be used to allow full access to the new database.
6. A user authentication dialog box will then appear. Enter "Admin" and the password set earlier in FileMaker Pro. A user with "Admin" permissions for a specified database can modify permissions for other users of that database, including giving them "Admin" permission.
7. At this point, options are presented to assign database action privileges or any field-level setting. Links can also be followed to log in and view another database.

If administrators are established for specific databases, they can access their particular database by entering the database name in the initial prompt and then the username and password that has been established for them in the user authentication dialog. They will then be able to make changes to usernames, permissions, record-level or field-level access to that particular database. To get to another database they would need to know the password for it, so other databases are protected.

Note: Remote administration of Lasso Security will not work with versions of WebSTAR prior to 2.0.

Note: Do not edit any of the HTML files located within the security folder (except for customizing security violation pages as indicated below) or change their names or locations.

Custom Security Violation Pages

There are two security violation error pages that can be returned to the end user, one for database-level security errors and another for field- and record-level security errors. These error pages are standard HTML pages placed in the "Security" folder and are given the following names:

Name	Purpose
Database_Violation.html	Error response returned to user for database-level security violations
Field_Violation.html	Error response returned to user for field-level and record-level security violations

The "Security" folder should be located at the root level of your Web serving folder. If Lasso doesn't find these files or is unable to open them for any reason, default security violation pages (internal to the Lasso application) are returned to the end user.

Important: The violation error pages can be customized to display your own message or images; however, do not change their names or locations.

Creating Links to Detail

When using "exactsearch," "exactupdate," or "exactdelete" to protect specific records with Lasso security, detail links must contain the username and password information in order for a search to be completed securely. Detail links can be constructed manually as follows:

1. Include the -lassousername and -lassopassword tags set to the Admin username and password.
2. Include all of the fields that are required for the exact search and build the elements of the detail link without using the [detail_link: …] tag. For example, if an exact search is set to the fields "id" and "name," create the link as follows:

   ```
   <a href="action.lasso?-database=database_name&-layout=layout_name&
   -recid=[recid_value]&
   -operator=eq&amp;id=[field:"id"]&-operator=eq&amp;name=[field:"name"]&
   -maxRecords=all&-search">
   Link to Detail</A>
   ```

   Note the use of the "equals" operator.

   The image tag must also be constructed manually, for example:

   ```
   <img src="action.lasso?-database=database_name&
   -recid=[recid_value]&photo=jpeg&
   -operator=eq&amp;id=[field:"id"]&-operator=eq&amp;name=
   [field:"name"]&-image">
   ```

Note that the above ampersands used before field names are encoded as "&" to prevent the ampersand from completing an HTML character reference.

Although the user will see the special field values for their record displayed in the location field, users will only see the values they entered. These values will also persist only for the length of the session. No one else will be able to gain access to the values in those special fields (unless they are standing over your shoulder watching you type in the values). In addition, you could have an "exit" page that would flush the cached pages using the Lasso [header] tag.

One could also put the detail link into a form so that a submit button is selected to go to the link. The advantage here is that the form can be set up as "post" so that the link will not display the location, nor the search args string containing the values for the "exactsearch" fields.

Note: Because of the way that FileMaker Pro indexes a field, an exact match is a match of a single "word" in a field, and not the value for the entire field. This is because FileMaker Pro indexes all fields by breaking up the field contents into separate words. This is important to note when using equals matches to find specific usernames or passwords. To overcome this, you could restrict users to a single word with no spaces for the field. Another option would be to either change the field to a "number" type field (the comparison would then be only on the first indexed item), or change the "Storage Options" to "index as ASCII" (this is found in the "Define Fields" dialog and will require that the password be case-sensitive). See CHAPTER 8: SEARCHING for more details.

Password Protect Individual Records

What follows is a description of how to password protect individual records:

1. Create two fields, "username" and "password" (they can be given any name) in the database to contain the protected records. Within FileMaker Pro, set these fields to contain required values.
2. Setup an entry in the "Lasso_Security.fp3" database with "exactsearch" applied to these fields.
3. Quit and restart Lasso.acgi, Lasso Server, or the Web server application (if using the Plug-in).

4. In a search page include text entry fields for "username" and "password" and a search button. Add an operator for each field to search using only the "equals" operator.

Here is an example login form:

```
<form action="action.lasso?-search&-database=resume&
-layout=main&-response=resume.html" method="post">
<p><input type="hidden" name="-operator" value="equals">
<input type="text" name="username" size="15">
<p><input type="hidden" name="-operator" value="equals">
<input type="text" name="password" size="15">
<input type="Submit" name="-search" value="Login">
</form>
```

Lasso and Realm-Based Security

Lasso Security and Web server realms-based security can work in conjunction, as well as in conflict with each other, depending on your configuration. Problems may arise if one accesses a Lasso format that relies on Lasso's security mechanism but is stored within a Web server password-protected realm and the username/password settings differ between the two security methods.

To avoid conflicts, you may simply choose to avoid using Web server realms and instead rely on protection by unique file suffixes. If this is inconvenient, it is not too difficult to design a setup where Lasso Security and Web server realms work in harmony. This is possible given that Lasso passes Web server information with each request. As long as settings between the two match for any given access request, there is no conflict.

Lasso loads realm information at startup and stores it in RAM. If realms are changed, Lasso should be restarted for the changes to take effect.

Note: At the time of this writing, there are issues which prevent proper interoperability of realms between Quid Pro Quo 2.0 and Lasso 2.5. These issues are not present with versions prior to Quid Pro Quo 2.0.

Note: Lasso Server does not use realm security; rather, files and folders can be protected using the powerful "processed file" options. See CHAPTER 6: LASSO SERVER for details.

Prompt for User Authentication

At some point it may be desirable to prompt for new authentication values to replace the current settings saved in the Web browser. The [header] tag can be used to "reset" the username and password by changing the HTTP header of the HTML file. Here is an example of the Lasso syntax used to accomplish this:

```
[header]
HTTP/1.0 401
Server: Lasso/2.5
WWW-Authenticate: Basic realm=some_realm_name
[/header]
To indicate that the authentication values stored in the Lasso_Security.fp3
database be used you would substitute some_realm_name with "database
name_of_database" as follows:
[header]
HTTP/1.0 401
Server: Lasso/2.5
WWW-Authenticate: Basic realm="database name_of_database"
[/header]
```

Separate the test "database" from the name of the database "name_of_database" with a space. Also, change "name_of_database" to the actual name of the database you are checking against (including "All Databases"). Put two returns before [/header] and change "some_realm_name" to an actual value which is the new realm (for example the folder your Lasso format files are in). The syntax is simply placed anywhere on the format file, for example at the top of the page (and can even be before the opening opening <HTML> tag). This will invoke a prompt for a username and password for the specified realm. See CHAPTER 16: HTTP CONTENT AND CONTROLS for more details.

Chapter 8: Searching

Lasso provides several options for refining a database query. The search parameters include: the fields that should be searched on, how to compare these values to the records contained in the database, the maximum number of records to return, how the returned records should be sorted, and whether any fields are required for the search to be submitted. In addition, the hitlist response page can display or utilize the previous search parameters.

Background Information

Search parameters are the components of a search that determine which fields are to be searched on and how the search is to occur. Although Lasso sends the search parameters to FileMaker Pro, it is the FileMaker Pro application which performs the actual search. Thus, the rules used for searching are set by FileMaker Pro. Lasso has been designed to emulate FileMaker Pro searches. For example, "begins with" is the default operator for both Lasso and FileMaker Pro. Also true for both is that a search with no search criteria will result in a "No Records Found" error, as opposed to finding all records, as is the case with other types of database applications.

FileMaker Pro Field Indexing

The rules used by FileMaker Pro in searching a database affect Lasso searches. This is most evident with how FileMaker Pro indexes a field, since indexing governs the way search parameters are compared to values in a field.

It is possible to view a field's index to see how this occurs (only for indexed fields). To view the index, enter "Browse" mode and place the cursor into the field. Go to the "Edit" menu and select "Paste Special," and then "From Index." You can then check the option "Show individual words" to see how the indexing has produced a list of words.

Note: Indexing will remove certain characters, such as periods (".") or "@" symbols and break values into components, separated by spaces.

An example of how FileMaker Pro responds to search requests sent by Lasso can be seen in Lasso operators' interpretation. An "equals" search will find those records that exactly match the word or string entered into the search field on a Lasso Web-based search form.

However, the comparison is applied to all indexed items or words in the field, not the field as a whole. This is because FileMaker Pro indexes all fields by breaking up field contents into components based on where spaces occur. In other words, if a field had "Tom and Jerry go fishing" an "equals" search would find "Tom," "Jerry" and "fishing," as well as "Tom and Jerry," since the index contains each word as separate items. Similarly, a "begins with" search finds all records containing the first characters in every individual word. Thus, if the value "Frank" is entered into a "begins with" search, the records with "Franklin," "Ben Franklin" and "Hot Dogs and Frankfurters" would all be in the found set.

To overcome FileMaker Pro indexing, indexing could be turned off for a field, but it will slow down searching dramatically. To turn a field's indexing on or off in FileMaker Pro, open the "Define Fields" dialog, select the field's options, and click the "Storage Options" button. Fields which are able to be indexed can be manually indexed by turning the "Indexing" radio button on. Indexing will occur when a find request is completed on that field.

Another option is to change the field type from "text" to "number." FileMaker Pro will then only compare the search parameter with the first value in the field (not each word). However, this works only when the search parameter is a single word. In addition, a field could be set to "Index as ASCII" (under "Storage Options"). When indexed as ASCII, all characters are used in the search; however, the field is also case-sensitive.

FileMaker Pro Search Operators

The following FileMaker Pro search operators can be used directly within Lasso search form fields:

>	(less than)	...	(range)
>=	(less than or equal to)	//	(today's date)
<	(greater than)	!	(duplicates)
<=	(greater than or equal to)		

Entering a FileMaker Pro search symbol into a field in a Lasso search form is the same as performing a Find operation directly in FileMaker Pro.

FileMaker Pro operators can also be included within a selection list. For example:

```
Number of Rooms: <select name="rooms">
   <option value="=Studio">Studio
   <option value=">0" selected>All
   <option value="1...3">1 to 3
   <option value=">2">More than 3
   <option value=">3">More than 4
</select>
```

Note that the double quotes used to enclose literal text in FileMaker Pro find operations cannot be used. Double quotes within double quotes will not work in HTML.

Lasso Search Operators

The -operator command tag, which can also be abbreviated as -op, directs Lasso on how to find matches. The operator is inserted directly before the field the operator affects. For example:

```
Search Values that: <select name="-operator">
   <option selected> contains
   <option> equals
   <option> not equals
   <option> begins with
   <option> ends with
   </select>
<input type="text" size=30 name="field_name">
```

All of the following operators are supported by Lasso:

Long Form	Short Form
equals	eq
not equals	neq
contains	cn
begins with	bw
ends with	ew
greater than	gt
greater than or equals	gte
less than	lt
less than or equals	lte

Note: Values used with the -operator command are not case-sensitive.

It is not necessary to have the user select an operator in the search form, as the default operator is "begins with." The operator will default to "begins with" for each field that is not immediately preceded with an operator.

As with selection lists, you can display alternate text for comparison operators, if the attribute "value" (e.g., value="contains") is used. For example:

```
Search Values That: <select name="-operator">
      <option value="equals" selected> =
      <option value="greater than"> &gt;
      <option value="greater than or equals"> &gt; or =
      <option value="less than"> &lt;
      <option value="less than or equals"> &lt; or =
      </select>
<input type="text" size=30 name="field_name">
```

The values specified must match either the "long form" or "short form" (abbreviated) of the search operators supported by Lasso. An example of an operator using abbreviations is as follows:

```
Search Values That:<select name="-operator">
      <option value="bw" selected>begins with
      <option value="eq">equals
      <option value="cn" >contains
      <option value="ew">ends with
</select>
<input type=text size=30 name="YourFieldName">
```

The -operator command may also be used as a hidden input field to "hard-code" a search. For example:

```
<input type="hidden" name="-operator" value="contains">
```

Date and Time Search Operators

If a field is defined as a date or time type field, then you must specify an operator that is either "equals," "greater than," "greater than or equals," "less than," or "less than or equals." For example:

```
<p align=right><b>Hire Date</b>
<select name="-operator">
   <option value="gte"> from
   <option value="gt"> after
</select>
<input type=text name="Hire Date" size=10>
<select name="-op">
   <option value="lte"> to
   <option value="lt"> before
</select>
<input type=text name="Hire Date" size=10>
```

For an [inline: …] tag to search for a date value in a date field, the following could be used:

```
[inline: database="site_log.fp3", layout="Main", operator="eq",
"Date"=server_date, search]
```

Logical Operator

A logical operator (AND or OR) can be used to determine whether the search criteria will find:

- Records that fulfill all the parameters indicated (this would be an AND-type search).
- Records in which any one (or more) of the parameters indicated are valid (this would be an OR-type search).

The logical operator applies to all search fields in the form. If no operator is indicated, the default logical operator is AND. The syntax for selecting a logical operator is as follows:

```
<input type="radio" name="-logicalop" value="and" checked>AND
<input type="radio" name="-logicalop" value="or">OR
```

This can also be coded as a hidden input type. For example, to specify OR-type searching, use:

```
<input type="hidden" name="-logicalop" value="or">
```

Multiple Logical Operators and NOT Support

The -opbegin and -opend command tags can be used to create a field-level operator, or multiple field-level operators. Any fields specified between the -opbegin and -opend tags in a search form will be grouped under the specified logical operator. The value of this tag can be "and," "or," or "not." The default is "and." There must be one -opend tag for every -opbegin tag. The -opend tag does not need to specify any value.

Here is one idea for how this can be used:

Search for this …

```
Model: <input type="text" name="model" size="30">
```

and, either

```
<input type="hidden" name="-opbegin" value="or">
  Price: <input type="text" name="Price" size="30">
```

or

```
    Color: <input type="text" name="color" size="30">
<input type="hidden" name="-opend">
```

The "not" logical operator can be applied only to one subsequent item to indicate that the value entered should not be used in a search. The item can be either a search field, or another -opbegin tag which can group other search fields or -opbegin tags. For example:

```
<input type="hidden" name="-opbegin" value="not">
   <input type="hidden" name="-opbegin" value="and">
      Do not include Names: <input type="text" name="name" size="30">
      Do not include Dates: <input type="text" name="date" size="30">
   <input type="hidden" name="-opend">
<input type="hidden" name="-opend">
```

In the example above, both input values contained within the nested "and" operator are required in order for the "not" operation to succeed.

If the [search_args] … [/search_args] tags are used on the reply page, then the "no_ops" parameter should be used as follows:

```
[search_args,no_ops] ... [/search_args]
```

The current field-level logical operators will not be displayed along with the other fields used in the search.

Sort Field and Order

Search results can be sorted using the -sortfield and -sortorder command tags.

- **-sortfield** indicates which field (or fields) to sort.
- **-sortorder** specifies how the search field (or fields) should be sorted.

The optional -sortfield tag instructs Lasso which fields sort the results. Any number of fields can be included (as long as they appear on the referenced FileMaker Pro layout). Records are left unsorted if the -sortfield command is not used, or is not given a value. The value of "unsorted" (not case-sensitive) is recognized by Lasso to leave records unsorted. An example of the -sortfield command is as follows:

```
Select field to sort by: <select name="-sortfield">
      <option selected>Unsorted
      <option>First Name
```

```
<option>Last Name
<option>Employee Number
<option>Hire Date
<option>Group
<option>Shift</select>
```

To have several fields used in a sort, list multiple -sortfield tags in the format file. The sorts are nested in the order in which they appear in the HTML. In other words, the first one becomes the primary sort and the next is the secondary sort and so on. For example, if the primary sort is on the "city" field and the next is on the "last name" field, the records returned would be sorted by the city field, and then, within groups of cities, the records would be sorted by the last name.

Multiple sorts can also be accomplished using a scrolling list and holding down the "command" key to make multiple field name selections, or by using a checkbox selection list. In these cases, the primary field sorted is the first item selected, the secondary is the one that was selected second, and so on.

If the sort order is indicated, the -sortorder command applies to the -sortfield tag that appears immediately before it. It is reset to "ascending" for subsequent sort fields. The -sortfield command tag is optional.

The values of "ascending," "descending," or "custom" (not case-sensitive) are recognized by Lasso to determine how to sort the results of a search. The default sort order is ascending if none are indicated.

Custom sorting is identical to the FileMaker Pro sort option "Custom order based on value list" using the value list that is specified for that field in the referenced layout (thus the field must have a value list). Any option can be included as a hidden field to predetermine the sort order. An example of the -sortorder command is as follows:

```
Sort By:<select name="-sortorder">
          <option selected>ascending
          <option>descending
          <option>custom
        </select>
```

Maximum Records

The -maxRecords command is used to indicate the maximum number of records returned per group in a hitlist page. Values may be any whole number. A value of "all" will return all records in the found set. The syntax for providing the visitor a choice of how many records to return per page is as follows:

```
Return  <select name="-maxRecords">
            <option>5
            <option selected>10
            <option>20
            <option>all
        </select> records per page
```

The number of records per group can also be predetermined using a "hidden" HTML input field.

Find All Records

The Lasso action -findall is used to find all of the records in the database. For example:

```
<input type="submit" name="-findall" value="Find All">
```

Or, as part of the form action:

```
<form action="../Lasso.acgi?-database=YourDBName&
-layout=YourLayoutName&
-response=Pathto/YourFileName.html&-findall" method="post">
```

The -findall action uses a default of 50 maximum records, so if all records are to be found, indicate -maxRecords="all" within an embedded URL or within the form.

The -findall action can also be specified in an embedded URL which can be used in a hyperlink as follows:

```
<a href="Lasso.acgi?-database=YourDBName&-layout=YourLayoutName
&-response=YourFileName.html&-maxRecords=all&-sortfield=YourFieldName
&-sortorder=ascending&-findall"> View all records in Database</a>
```

In order to execute the -findall action, the user must have "search" permission for the database. If the user doesn't have search permission, a database security violation error is returned. If any field in the database has been defined in the Lasso Security database with an "exactsearch" restriction, a security violation will be returned if the user tries to execute the -findall action.

Display Search Parameters

The search parameters used in performing a search may be displayed as HTML. Search parameters are sometimes useful to display on the hitlist return page or any subsequently retrieved pages from the initial search. There are four search parameters that can be displayed:

[search_field] and **[search_value]** — These display the name of the field search and the value entered for the search. These tags accept the optional "raw" parameter which prevents Lasso from applying HTML character encoding (for example, [search_value,raw]). The "url" parameter can also be used to specify URL encoding. If omitted, HTML encoding is performed by default.

[search_op] — This displays the operators applied to the field. This tag accepts optional parameters "short" and "long," as in [search_op, short]. The default is "long." "Long" inserts the field operator in its long form: "equals," "begins with," "greater than," etc. "Short" uses the Lasso abbreviations "eq," "bw," "gt," etc. Specify "short" if using [search_op] to create a URL.

[logicalop_value] — This substitutes "and" or "or" into the output HTML based on the logical operator specified in the original search.

The tags [search_field], [search_op], and [search_value] are valid only within a [search_args]…[/search_args] container. The tag [logicalop_value] can be used anywhere, since the logical operator applies to the search as a whole, and cannot be applied to specific fields.

All HTML appearing between [search_args] and [/search_args] containers is repeated for every field included in the search. The tags are simply placed in the reply file as follows:

```
[search_args]
    [search_field]
    [search_op]
    [search_value]
[/search_args]
[logicalop_value]
```

As with all other Lasso tags, HTML can be used to format the returned values.

Generally, these tags are useful only for displaying the search arguments. To include these values in a conditional [if: …]…[else]…[/if] expression, use the [form_param: …] tag discussed elsewhere.

[search_args]…[/search_args] — Substitutes the search criteria used in the previous search. The following sub-container tags are valid only within the [search_args] container, and one or all can be used:

[search_field] — Will display the name of the field the search was made on

[search_value] — Will display the value entered for the search for a particular [search_field].

[search_op] — Will display the operators applied to the [search_field]. [search_op] accepts the optional keywords "short" and "long," as in [search_op,short]. The default is "long" which inserts the field operator in its long form: "equals," "begins with," "greater than," etc. "Short" uses the Lasso abbreviations "eq," "bw," "gt," etc.

Note that everything between the [search_args]…[/search_args] and **[sort_args]…[/sort_args]** containers will be repeated for every field used for the search. For example:

```
[search_args]
   Field Searched: [search_field]<br>
   Operator: [search_op] <br>
   Value Entered: [search_value] <br><hr>
[/search_args]
```

Complex Embedded Searches

When a found set of records is called up, alterations to the displayed hitlist may be desired. However, to resort or reformat the list, the search must be performed again. Instead of having the user return to a search form and resubmit their search, links using all of the custom search settings selected by a user on the previous page can be created.

The previous search elements can be retrieved using the following tag construction:

```
<a href="action.lasso?-search&-database=[database_name,url]&
-layout=[layout_name,url]&-response=[response_file_path,url]
[search_args]&-operator=[search_op,url]&[search_field,url]=[search_value,url]
[/search_args][sort_args]&-sortfield=[sort_field,url]&
-sortorder=[sort_order,url][/sort_args]&-logicalop=[logicalop_value]&
-maxrecords=[maxrecords_value]&-skiprecords=[skiprecords_value]">
Link to Accomplish the Same Search </a>
```

For example, a link could be set up to find a set of records that are part of the current found set, but sorted in a different order, as follows:

```
<a href="action.lasso?-sortfield=YourFieldName&-sortorder=ascending&
-database=[database_name,url]&-layout=[layout_name,url]&
-response=[response_file_path,url][if:lasso_action=="findall"]&-findall[else]&
-search[search_args]&-operator=[search_op,url]&
search_field,url]=[search_value,url][/search_args]&
-logicalop=[logicalop_value][/if]&-maxrecords=[maxrecords_value]&
-skiprecords=[skiprecords_value]">Search Again and Sort by Date </a>
```

Note that there should be no spaces or carriage returns anywhere in this URL string. The URL keyword was used to URL-encode field values.

The above embedded URL link was constructed by first specifying new sort criteria and then instructing Lasso to substitute values from the previous search:

```
action.lasso?-sortField=change_field_name_here&-sortorder=ascending&
-database=[database_name,url]&-layout=[layout_name,url]&
-response=[response_file_path,url]&...
```

Since there can be multiple instances of fields used in the search, the [search_args]…[/search_args] container is used to ensure each item is included:

```
...&-search[search_args]&
-operator=[search_op,url]&[search_field,url]=[search_value,url][/search_args]...
```

Because no search arguments are used on a -findall action, a conditional statement is used to exclude the [search_args: …]… [/search_args] container, as well as the -logicalop tag:

```
...[if:lasso_action=="findall"]&-findall[else]&-search[search_args]&
-op=[search_op,url]&[search_field,url]=[search_value,url][/search_args]&
-logicalop=[logicalop_value][/if]...
```

Required Field Entry on Form Submit

The -required command tag is used to require that a value is entered into a specified field before a form can be successfully submitted. Although -required can also be used on "add" or "update" forms, it is mentioned here since this is where it has the

greatest utility. For example, if a search form has fields for a username and password the -required tag could be used to require that specific values be entered into both fields.

The -required tag is placed in a hidden input field and applies only to the field that immediately follows. For example:

```
<input type="hidden" name="-required">
<input type="text" name="fieldname" size=20>
```

The -required command can also be used in combination with operators. For example, the required field could be formatted as follows:

```
<input type="hidden" name="-required">
<input type="hidden" name="-operator" value="eq">
<input type="text" name="fieldname" size=20>
```

Different -operator or -required commands can be specified for several different fields on the same form.

Any form that was submitted without values entered into the "required" field is rejected. In this case, Lasso displays a "required field missing" error page. A customized error page can be used in place of this generic message. The required field error page is specified with the -reqfieldmissing tag (for both search and add forms). The customized error is specified as:

```
<input type="hidden" name="-reqfieldmissing"
value="InsertPathtoFile/FileName.html">
```

Chapter 9: Graphics and Multimedia

Lasso is able to display graphic images in two ways: either from images stored in FileMaker Pro, or from images stored and referenced in the Web server folder. In the first case, the [image: …] tag can be used to convert PICT files stored in FileMaker Pro to either JPEG or GIF. In the latter case, GIFs and JPEGs stored within the Web server folder may be referenced using standard HTML and combined with textual content retrieved on-the-fly from your database.

Serving Images Stored in FileMaker Pro

Lasso has the ability to directly convert FileMaker Pro pictures for display in Web browsers. Graphics are stored in a FileMaker Pro container field as a PICT file. The FileMaker Pro field which contains the PICT image is specified in a Lasso format file using the [image: …] tag. The format used is as follows:

```
<img src="[image: FieldName, gif]"> <img src="[image: FieldName, jpeg]">
```

The default parameter is "jpeg." If "gif" is indicated, you must have "clip2gif" installed as described below.

Additional parameters can be used with the "jpeg" parameter to control the amount of compression. Specify 16 or 32 as the bit-depth for each pixel. The default is 16. Specify a number 0 through 4 for the compression amount. 0 yields the best quality, but results in larger files. Four yields satisfactory quality for smaller files. The default is 4.

Some examples:

```
[image: FieldName, jpeg, 16, 0]
[image: FieldName, jpeg, 32, 4]
[image: FieldName, jpeg, 16, 4] (equivalent to [image: FieldName, jpeg])
```

Warning: Setting the bit-depth to 32 will greatly increase Lasso's memory requirements! Increase the memory allocation of the Lasso application if you plan to use 32-bit JPEG images. If using the Plug-in, increase the memory of the Web server application.

When a Lasso action is called, the picture is converted to GIF or JPEG. When the format file is processed, Lasso displays the image by substituting the [image: …] tag with HTML code that points to the record where the image is located. The syntax is as follows:

```
<img src="/Lasso.acgi?[database]=database_name&[recid]=#&
photo=jpeg&[image]">
```

In the above example, the symbol "#" is the record ID for the targeted image and "photo" is the field name. If, instead of the converted image displaying after Lasso processes the page the actual HTML code for specifying the image conversion appears, then an error has occurred. If the image is not displayed, it could be because not enough memory is available, no parameter was indicated, the image field was not correctly specified, the "gif" parameter was specified and clip2gif was not installed, or because there was no image in the database to return.

Implementation Notes

If you plan to return images from a FileMaker Pro database, consider several points:

- Increase Lasso.acgi's allocated memory to at least 1500K, or to 2500K if converting 32-bit JPEG images. (the more RAM the better). If using the Plug-in or Lasso Server, allocate more memory to the server application.
- To serve JPEG images directly from the FileMaker Pro database using the [image: …] tag, QuickTime is required.
- If using the [image: …] tag to return PICT graphics, then use FileMaker Pro 3.0v4 or greater. Updates can be obtained from Claris Corporation.

GIF Conversion Using clip2gif

JPEG format generally works best for photographs or other photo-realistic images, while GIF format generally works best for pictures that contain large blocks of solid colors such as logos or simple graphics. GIF format is also supported by all browsers (except text-only browsers). In order to convert FileMaker Pro pictures to the GIF format via Lasso, the freeware utility clip2gif must be installed on your Mac OS server. Using the "gif" parameter, pictures will automatically be transferred to an intermediary application, clip2gif, when a Lasso action is called. clip2gif converts the image to GIF.

Additional parameters can be used with the "gif" parameter to control the quality of the image. Specify "true" or "false" for interlaced or non-interlaced; the default is "false" (non-interlaced). Also for GIF images, specify 1, 2, 4, or 8 for bit-depth; the default is 8. The bit-depth determines how many colors are used in creating the GIF. For example, a black and white image is considered a 1-bit, 2-color image.

Some examples:

```
[image: image_YourFieldName, gif, false, 8]
[image: image_YourFieldName, gif, true, 4]
[image: image_YourFieldName, gif, false, 2]
```

The clip2gif application does not need to be launched and left open, it just needs to be on the same drive. However, the most reliable method is to place clip2gif in the same folder as Lasso. As of December, 1997, the current version of clip2gif is 0.7.2. You can download clip2gif from your favorite Macintosh software archive or the clip2gif home page at:

```
http://iawww.epfl.ch/Staff/Yves.Piguet/clip2gif-home/
```

Note: Please see the clip2gif copyright notice at the back of this manual.

Extending the [image: ...] Tag

The field containing the image should not be placed on the layout referred to in a search as this will dramatically slow down the search operation.

It is not possible to check an image field to determine whether an image is available using an [if: …] conditional check. For example, the following cannot be used to determine if an image is present:

```
[If:field:"Photo"==""][else]
<img src="[image: "Photo"]" align=left>[/if]
```

As a workaround, create an additional field in the database and for each record that has an image, check a box. An [if: …] comparison can then be made on that field to determine whether the image tag is used. For example, for a field called "image_available," include the following:

```
[If:field:"image_available"==""][else]
<img src="[image: "Photo"]" align=left>[/if]
```

Displaying Images from the Web Serving Folder

Images can be displayed directly from the Web serving folder using standard HTML. For example:

```
<img src="path-to-image/image.gif" border="0">
```

To specify an image for a record, create a field in a database that contains only the name (or path and name) of the image. The syntax to reference this field would appear as follows:

```
<img src="images/[field:"imageFieldName",url]" border="0">
```

In the example above, a folder within one's Web serving folder named "images" contains GIFs or JPEGs. The exact names (text only) of these images are stored within a database field named "imageFieldName." For a value named "picture.gif" stored within the database field "imageFieldName," the previous example would generate the following result:

```
<img src="images/picture.gif" border="0">
```

When the server returns the HTML to the browser, the image is displayed (as long as the proper path name to graphics stored within your Web server folder has been specified).

When using images as path names stored within FileMaker Pro fields, use the "url" parameter for the appropriate field. The "url" parameter instructs Lasso to perform URL encoding rather than HTML encoding (the default). For more details on character encoding, refer to APPENDIX B: CHARACTER SETS AND TRANSLATION.

Rotating Banners Example

To incorporate an image that changes each time a Web page is called up, it is possible to have Lasso randomly return a record containing a different image. To accomplish this, first save various images into a database. Then, substitute the name of your database, layout, and the field that contains the image into the following syntax:

```
[inline: database="YourDBName", layout="YourLayoutName", random]
   [If: inline_result=="noErr"]
      <img src="[image: "YourFieldName",jpeg]">
   [else]ERROR= [inline_result]
   [/if]
[/inline]
```

Place this in the location where the image is to display.

The inline action occurs whenever the page is processed (including when the page is reloaded). The optional conditional statement is used to check for errors. To ensure that something appears, remove the error message (between [else] and [/if]) and add a default image reference. Also, make sure that the inline action occurs on a layout that does not actually contain any images, as this will slow the search operation.

This procedure also works if all the images are saved as JPEG or GIF images in a folder within the Web serving folder. Simply substitute the image reference with a field value from the database that indicates the location of a specific JPEG or GIF image. The syntax is as follows:

```
[inline: database="YourDBName", layout="YourLayoutName", random]
<a href="http://[field:"InsertFieldName",url]">
   <img src="path-to-image/[field:"InsertFieldName",raw]" border=0></a>
[/inline]
```

Note that HTML is also included to make the image a linked URL reference, and that "path-to-image" should be replaced with the path relative to the active URL.

Counting the Number of Times an Image is Displayed

To know how many times an image was displayed, add a new field to the image database, for example "visitcount" and add a nested inline statement to increment that field by one each time it is found. For example:

```
[inline: database="YourDBName", layout="YourLayoutName", random]
   [If: inline_result=="noErr"]
      <img src="[image: "YourFieldName",jpeg]">
      [inline: update, database="database_name", layout="layout_name",
visitcount=(math-add:field:"visitcount",1), recid=recid_value] [/inline]
   [else]ERROR= [inline_result]
   [/if]
[/inline]
```

Because the nested inline may slow things down a bit, you may want to use a post-inline tag. The syntax for the page displaying the banner would then be as follows:

```
[inline: database="YourDBName", layout="YourLayoutName", random]
   [If: inline_result=="noErr"]
      <img src="[image: "YourFieldName",jpeg]">
[post_inline: post_response="inline_imagecount.txt", "which"=recid_value,
```

```
"visitcount"=(math-add:field:"visitcount",1)]
   [else]ERROR= [inline_result]
   [/if]
[/inline]
```

The inline update action would occur when Lasso next had idle time. The named parameter "which" does not have any meaning as a Lasso tag or parameter; it is simply a word that is used in conjunction with the [form_param: …] tag. The record ID value can then be retrieved on the page processed by the [post_inline: …] tag using [form_param: "which"].

The file that the post inline processes, "inline_imagecount.txt," contains the statement:

```
[inline: update, database="YourDBName", layout="YourLayoutName",
visitcount=form_param:"visitcount", recid=form_param:"which",
token=form_param:"which"] [/inline]
```

As a way to test if the post inline is occurring as expected, the [log: …] tag could be added within the inline on the "inline_imagecount.txt" file. To do so, add the following statement within the [inline: …]…[/inline] container:

```
[If: inline_result=="noErr"]
   [log:window]Record was updated: [form_param:"visitcount"]
      times for record ID: [token_value]
   [/log]
[else]
   [log:window]ERROR= [inline_result]
   [/log]
[/if]
```

The result of the inline will be displayed in the log window. Note: A token was used in the inline statement in order to pass the record ID value ([form_param: "which"]) to the log tag contained within the inline.

By extending this example, the image that has been shown the fewest number of times could be shown rather than a random image. For example:

```
[inline: database="YourDBName", layout="YourLayoutName",
   sortfield=visitcount,
   sortorder=descending, findall]
   [inline: database="YourDBName", layout="YourLayoutName",
recid=recid_value, search]
   <img src="[image: "YourFieldName",jpeg]">
   [/inline]
[/inline]
```

The first inline searches the entire database for the record least often displayed. Note that the -skiprecords tag could also be used to skip the first one or two records. The second inline searches for one specific record, in order to retrieve only one image from the database.

Serving Multimedia Files

Lasso can serve multimedia files, such as AIFF and WAV sounds or QuickTime movies, in the same way Lasso can serve images: by file and path name references stored within a database field. Additional HTML syntax for dealing with sound:

```
[if: "sound_file"==""] [else] <embed src="path_to_sounds_folder/
[field:"name_of_field"]" width=2 height=2 autostart=true loop=true>
<bgsound src="path_to_sounds_folder/[field:"name_of_field"]> [/if]
```

The first "embed" tag is for Netscape Navigator, the "bgsound" tag is for Microsoft Internet Explorer. Any type of supported multimedia file can be served by storing its file and path name reference in a FileMaker Pro field and returning that information in the output HTML.

Chapter 10: Repeating Fields and Related Fields

Repeating Fields

Adding Records to a Repeating Field

Adding records to a repeating field in FileMaker Pro requires the following:

- Make sure the targeted FileMaker Pro layout displays all the repetitions specified for the field.
- Display the repeating field's specified number of repetitions within one's format file by simply placing multiple instances of an HTML text input reference to the field. For example, a repeating field set to three repetitions is properly referenced as follows:

```
<input type="text" name="fieldname" value="" size=30>
<input type="text" name="fieldname" value="" size=30>
<input type="text" name="fieldname" value="" size=30>
```

Displaying Repeating Fields

When displaying a repeating field, the number of repetitions defined for that field in the specified layout are returned. The following syntax is used in any post-Lasso format file to display the contents of a repeating field:

```
[repeating: fieldName]
   [repeat_value]
[/repeating]
```

Updating Repeating Fields

The following syntax is used in any post-Lasso format file to update the contents of a repeating field:

```
[repeating: "fieldName"]
   <input type="text" size=30 name="fieldname" value="[repeat_value]">
[/repeating]
```

Repeating fields can include value lists using the [value_list] and [option] tags inside a [repeating: …] container as follows:

```
[repeating: "fieldName"]
   [option: "fieldName"]
[/repeating]
```

Related Data

FileMaker Pro allows for database files to be related together to share data. Fields are related either by a one-to-one or one-to-many relationship. In a one-to-one relationship, the related databases share a common field, but only one record in each database is related to a record in the other database. One-to-many relationships are handled by a "portal" to display multiple related records from another database. FileMaker Pro does not typically handle many-to-many relationships.

Lasso exchanges data with related fields that appear either inside or outside of portals. Related records in portals can be updated and new related rows added to the portal. Deleting rows can be accomplished if the databases are specially designed to accommodate this.

Related Fields

Displaying Related Fields

Related fields may be specified anywhere in your Lasso format file (for related fields positioned outside of a portal within one's FileMaker Pro layout) or specified within the [portal: …] container tag (for related fields set within a portal in one's FileMaker Pro layout).

Adding and Updating Related Fields

A related field is identified by the name of the relationship along with the field name separated by double colons. For example:

```
[field: "InsertRelName::YourFieldName"]
```

To add to a related field, the following syntax is used:

```
<input type="text" size=30 name="InsertRelName::YourFieldName">
```

To update a related field the following syntax is used:

```
<input type="text" size=30 name="field: InsertRelName::YourFieldName"
value="[field: InsertRelName::YourFieldName]">
```

The related field must appear in the referenced database layout. The same Lasso rules for the [field: …] command apply to related fields. For instance, if a value list is assigned to the related field in the FileMaker Pro interface, then value lists can be dynamically generated on post-Lasso forms. In addition, character encoding parameters can be used. For example, the "break" parameter translates all carriage return characters to HTML
 tags. Parameters are used with related fields as follows:

```
[field: "InsertRelName::YourFieldName",break]
```

Portals

The [portal: …]…[/portal] container is used to display or update multiple related records contained in a portal in the referenced FileMaker Pro database layout.

The opening [portal: …] tag must specify the name of the FileMaker Pro relationship. In addition, the field names within a [portal: …] tag must also be fully specified with the FileMaker Pro "Relationship Name::Field Name" syntax, as follows:

```
[portal: "InsertRelName"]
   [field: "InsertRelName::YourFieldName"]
[/portal]
```

All HTML within a [portal: …] container is repeated once for each record in the portal's found set. Several field references (from the same relationship) may also be included within a [portal: …] container, as in the following example:

```
<b>Record for: [field: "contact"]</b>
<table border=1>
   <tr>
      <td rowspan=4 valign=middle><img src="sunny.gif" border=0
         alt="sunny">
      </td>
   </tr>
[portal: "travel"]
   <tr><td><b>Cities:</b></td><td>[field: "Travel::Cities"]</td>></tr>
   <tr><td><b>Countries:</b></td><td>[field: "Travel::Countries"]</td></tr>
   <tr><td><b>Airlines:</b></td><td>[field: "Travel::Airlines"]</td></tr>
[/portal]
</table>
```

Only related fields from relationships specified in the [portal: …] tag can appear within a [portal: …] container.

The [repetition] tag used within an [if: …] conditional statement can also be used within [portal: …] containers. The repetition can test for the number of related records (rows), and based on this, display any HTML element or Lasso-returned value in that pattern. For example, to shade every other row in a table, use the following:

```
<table>
<tr>  <td>Name</td>
<td>Rank</td>
<td>Serial Number</td></tr>
[portal:"related"]
<tr>  <td>[field:"related::name"]</td>
      <td>[field:"related::rank"]</td>
      <td>[field:"related::serial"]</td>
</tr>
[if: repetition=2]
<tr bgcolor="#dddddd"><td>[field:"related::name"]</td>
   <td>[field:"related::rank"]</td><td>[field:"related::serial"]</td>
</tr>
[/if]
[/portal]
</table>
```

The [value_list: …]…[/value_list] container may appear inside a [portal: …]…[/portal] container as follows:

```
[portal: Relationship Name] [field: Relationship Name::Field Name]
   [value_list: Relationship Name::Field Name][list_value][checked]
   [/value_list]
[/portal]
```

Example of Updating Related Records Within a Portal

The following example shows how to set up a portal with updatable related records set within table cells.

```
<p><form action="/Lasso.acgi?-recid=[recid_value]&-database =Employees&
 -layout=Summary&-update&-response= Example/view_expenses.html"
method="post">
[portal: "expenselist"]
<tr>
   <td valign="top">
   <select name="expenselist::expense_category">
      [option: "expenselist::expense_category"]</select></td>
   <td valign="top">
   <input type="text" size="30" name="expenselist::description"
      value="[field: "expenselist::description"]"></td>
```

```
    <td align="center" valign="top">
    <input type="text" size="8" name="expenselist::date"
        value="[field: "expenselist::date"]"></td>
    <td align="right" valign="top">
    <input type="text" size="10" name="expenselist::amount"
        value="[field: "expenselist::amount"]"></td>
</tr>
[/portal]
<tr>
    <td><input type="submit" value="update">
    <input type="reset" value="reset"></td>
    <td colspan=3 align="right" valign="top">
    total: [field: "total_expense"]</td>
</tr>
</form>
```

Deleting Related Records Within a Portal

To delete records in a portal, a unique serial number field must be established in the related database. This is because the record ID cannot be retrieved from a related database when the data from the related database is viewed through a portal. The record ID that is present is actually the record ID for the record in the main database. With FileMaker Pro 4.0, the record ID of the related database can retrieved using a calculation field and the "Status(CurrentRecordID)" calculation.

In the following example, a delete is accomplished by first searching the related "Employees_Related.fp3" database for the "expense_id" field to identify the record to be deleted.

```
<a href="action.lasso?-database=employee_related&-layout=main&
-response=expenses_view.html&-token=[field: employee_id]&
expense_id=[field: expenselist::expense_id]&-search">
```

The delete occurs after the reply file is processed via an inline action. The inline in the "expenses_view.html" file is as follows:

```
[inline: database=database_name, recid=recid_value,
    token=token_value, delete]
[/inline]
```

After the delete action is completed in the related database, a search must be carried out on the main database to call back the record you started with. For example:

```
[inline: database=Employees, layout=Summary,
employee_id=token_value, token=token_value, search]
    [include: "expense_include.txt"]
[/inline]
```

Since the record ID of the record in the main database cannot be used to find the current record, a token was set to the value in the "employee_id" field before linking to the related record. The "employee_id" field is the unique serial number field used to identify the record and is used here in lieu of the actual record ID. It is also the key value that relates the main "Employees.fp3" database with the "Employees_Related.fp3" database. Note that the record ID of the record in the main database could be used to serve the same purpose using token=[recid_value].

Chapter 11: Executing FileMaker Pro Scripts

For active Web servers, relying on FileMaker Pro scripts to assist with data management is not recommended. In the majority of cases, similar functionality is achieved using Lasso's rich data manipulation capabilities. As Lasso is multithreaded and FileMaker Pro is not, routines executed in Lasso will be much faster. For situations where executing FileMaker Pro scripts is required, Lasso provides several options.

Specifying a FileMaker Pro Script

There are two parts to specifying a FileMaker Pro script within a Lasso format file:

- Specify the -scripts action.
- Specify one of the six "FMP Script" commands (see below).

The syntax is as follows:

```
<input type="hidden" name="-doscript" value="InsertScriptName">
<input type="submit" name="-scripts" value="Run FileMaker Scripts">
```

An example embedded URL may appear as follows:

```
<a href="Lasso.acgi?-database=database_name&-layout=layout_name&
-response=Pathto/YourFileName.html&-doscript=Script_Name&-scripts">
Run Script</a>
```

FMP Script Commands

The six "FMP Script" commands define which FileMaker Pro script to execute and how the script is executed. These commands are described as follows:

- **FMP Script Post-Lasso (-doscript.post)** — Processes script after all other specified Lasso actions are completed. Brings FileMaker Pro to foreground while script is processing, then to background after completion.
- **FMP Script Post-Lasso in Background (-doscript.post.back)** — Processes script after all other specified Lasso actions are completed. Keeps FileMaker Pro in the background while processing.

- **FMP Script Pre-Lasso (-doscript.pre)** — Processes script before all other specified Lasso actions are completed. Brings FileMaker Pro to foreground while script is processing, then to background after completion.
- **FMP Script Pre-Lasso in Background (-doscript.pre.back)** — Processes script before all other specified Lasso actions. Keeps FileMaker Pro in the background while script is processing, then to background after completion.
- **FMP Script Pre-Sort (-doscript.presort)** — Processes script before Lasso -sort command is invoked in a Lasso -search action. Only works in conjunction with Lasso -search action. Brings FileMaker Pro to foreground while script is processing, then to background after completion.
- **FMP Script Pre-Sort in Background (-doscript.presort.back)** — Processes script before Lasso -sort command is invoked in a Lasso -search action. Only works in conjunction with Lasso -search action. Keeps FileMaker Pro in the background while processing.

Important Notes

- **Empty Found Sets** — If the FileMaker Pro found set is empty after performing the scripts specified for a -scripts action, Lasso returns a -50 error. The user should make sure the FileMaker Pro found set is not empty after performing a scripts action.
- **Security Concern** — Because of potential security concerns, the ability to execute FileMaker Pro scripts must be enabled with Lasso Security (see CHAPTER 7: SECURITY). In addition, if you allow users to upload Lasso format files and/or FileMaker Pro databases, beware of modifications that may compromise the security or performance of your Web server. You may want to require that any new databases that are uploaded to your Web server go through a quick security check to ensure the user has not inadvertently set a potentially destructive or processing-intensive script.

Chapter 12: Email

Lasso has the ability to automatically send an email message after a Lasso action is taken. The five-step code sequence for email notification can be included in any Lasso format file, embedded in a link, or used within an inline. The email is sent whenever the format file is submitted, the link clicked, or the inline processed.

In order to send an email, five email command tags must be assigned a value. The first four are as follows:

-emailhost — Sets the mail host (SMTP) through which the email is sent.

-emailfrom — Sets the sender's (from) email address. Only one address may be specified.

-emailsubject — Sets the subject that appears in the email header. Only one subject may be specified.

-emailformat — Sets the format file used to create the body of the email message. This format file can contain any of the usual Lasso formatting tags, and completely controls the format of the message. Only one format may be specified.

In addition to those four required tags, one of the following three must be specified, and the other two are optional:

-emailto — Sets the email address for the primary recipient.

-emailcc — Sets the email address for a recipient to receive a copy of the email message

-emailbcc — Sets the email address for a recipient to receive a "blind" copy of the email message.

The email message can include the contents of a single record, or all messages in the found set may be displayed using the [record] tag.

Lasso email command tags need to be customized to a specific email account (or get values from field values returned by Lasso). All email command tags included in the format file must be assigned a value or the message will not be sent.

Sending Email from a Form

The five Lasso command tags normally appear as hidden input parameters in an HTML format file using the following syntax:

```
<input type=hidden name="-emailhost" value="mail.senderdomain.com">
<input type=hidden name="-emailfrom" value="sender@senderdomain.com">
<input type=hidden name="-emailto" value="receiver@receiverdomain.com">
<input type=hidden name="-emailsubject" value="Email Delivered by Lasso">
<input type=hidden name="-emailformat" value="path_to_file/Email_Format.txt">
```

The optional tags are formatted in a similar manner:

```
<input type=hidden name="-emailcc"
value="ccreceiver@ccreceiverdomain.com">
<input type=hidden name="-emailbcc" value="bccreceiver@
bccreceiverdomain.com">
```

Lasso can send email to several recipients if several tags are included in the set of email tags. The tags that can appear multiple times on one form include:

-emailto, -emailcc, -emailbcc

For example, several additional recipients can be added as follows:

```
<input type=hidden name="-emailhost" value="mail.senderdomain.com">
<input type=hidden name="-emailfrom" value="sender@senderdomain.com">
<input type=hidden name="-emailto"
value="receiver1@receiver1domain.com">
<input type=hidden name="-emailto"
value="receiver2@receiver2domain.com">
<input type=hidden name="-emailto"
value="receiver3@receiver3domain.com">
<input type=hidden name="-emailto"
value="receiver4@receiver4domain.com">
<input type=hidden name="-emailsubject" value="Email Delivered by Lasso">
<input type=hidden name="-emailformat" value="path_to_file/
Email_Format.txt">
```

Sending Email with an Embedded URL

For an email message to be sent from an embedded URL, the five required email tags must be placed in a string with the Lasso action, for example:

```
<a href="Lasso.acgi?-database=YourDBName&-layout=YourLayoutName&
-response=path/response.html&-recid=[recid_value]&
-emailhost=mail.senderdomain.com& -emailfrom=
sender@senderdomain.com&-emailto=receiver@receiverdomain.com&
-emailsubject=Email%20Delivered%20by%20Lasso&
-emailformat=PathtoFile/Email_Format.txt&-search">Send Email </a>
```

The record ID reference "-recid=[recid_value]" is only necessary if the URL is on a detail page for a specific record and you want to continue to work with the current record (i.e. for updates or deletes).

An unrelated embedded URL could also be referenced as follows:

```
<a href="Lasso.acgi?-database=name&-layout=name&
-response=http://www.domain.com&-emailhost=mail.senderdomain.com&
-emailfrom=sender@senderdomain.com&-emailto=
receiver@receiverdomain.com& -emailsubject=Email%20Delivered%20
by%20Lasso& -emailformat=PathtoFile/Email_Format.txt&
-show">Send Email</a>
```

A different database and layout that contains the elements to be retrieved by the show action must be specified.

It is important to remember that, in any URL, all spaces should be removed and characters encoded according to the HTTP standard. It is not good practice, however, to use ampersands, slashes, or other characters reserved for URLs in database, layout, or field names. If a space is part of a database, layout, field name, or field value, the plus sign ("+") or the symbols "%20" should be used to represent a space. In addition, when a parameter is used, no spaces are used within the brackets (e.g., [field:job,raw]). The "url" parameter must be used with any database fields so the returned data is encoded for use in a URL (e.g., [field:fieldname,url]). Also, carriage returns cannot be used in the final embedded URL.

Sending Email within an [inline: ...]

An [inline: …] tag can also be used to send an email message without interacting with a FileMaker Pro database. Inline actions are processed on a post-Lasso reply page. The syntax appears as follows:

```
[inline: emailhost="mail.domain.com", emailto="recipient@domain.com",
emailformat="email.txt", emailsubject="insertsubject",
emailfrom="sender@domain.com", nothing]
[/inline]
```

Email Format File

An email message is formatted according to a format file. In addition to hard-coded text, as with other format files, email format files can contain Lasso substitution tags to display field values (e.g., [field: FieldName]). All Lasso format tags can be used in the email format file.

When a message is sent, Lasso will replace any field references with the data retrieved from the FileMaker Pro database.

A sample email format file follows:

```
The following record has been added to your Lasso Database:

Last Name: [field: last name]
First Name: [field: first name]
Social Security Number: [field: social security number]
Employee Number: [field: employee number]
Hire Date: [field: hire date]
Group: [field: group]
Shift: [field: shift]
Email Address: [field: email]
Home Page: [field:home page]
```

Sending Email Without a Database

It is possible to send an email message containing the data entered into a submission form, without interacting with a database.

In the example form below, notice the use of the -nothing action along with two email tags to be provided by the user.

Sample Submission Form

```
<html>
<head>
<title>Email Page</title>
</head>
<body>
<font size="6">Email Page</font>
<p>
<form action="action.lasso?-nothing" method="post">
<input type="hidden" name="-response" value="send.html"> (the response can
be any other file)

<input type="hidden" name="-emailhost" value="mail.host.com">
<input type="hidden" name="-emailto" value="receiver@receiverdomain.com">
<input type="hidden" name="-emailformat" value="email.txt">
```

```
Subject: <br>
<input type="text" size=30 name="-emailsubject"><p>

Your Email Address:<br>
<input type="text" size=30 name="-emailfrom"><p>

Make a choice:<br>
<select name="field1">
   <option>
   <option>1st choice
   <option>2nd choice
   <option>3rd choice
</select><p>

Check your values:<br>
<input type="checkbox" name="field2" value="value1">value1
<input type="checkbox" name="field2" value="value2">value2
<input type="checkbox" name="field2" value="value3">value3
<input type="checkbox" name="field2" value="value4">value4<p>

Comments:<br>
<textarea name="field3" rows=4 cols=40 wrap=soft></textarea>

<p><input type="Submit" name="-nothing" value="Send Email">

</font>
</form>

</body>
</html>
```

In the above example, the -emailsubject and -emailfrom values are entered by the user on the form.

Sample Email Format File

To include information submitted by the user on the previous form in an outgoing email message, a [form_param: …] tag must be placed within the email format file.

```
Here are the details of the form that was submitted [server_date] at
[server_time]:

1) [form_param:field1]

2) [form_param:field2]

3) [form_param:field3]
```

When the message is submitted, no action occurs on any database, but the email message is sent using the contents of the submitted form.

Note that the -required tag cannot be used for the -emailsubject and -emailfrom text entries, since the input is to a Lasso command, not a field. To have the input fields be required, either use a JavaScript, or create an [inline: …] tag on the reply page. Instead of the email occurring on the form page, it occurs on the reply using values provided by [form_param: …] tags.

Sample Using an Inline

To send an email within an inline without any interaction with a database, simply submit a form using the -nothing action and then use [form_param: …] tags on the reply page to retrieve the field values. For example:

```
<form action="action.lasso?-nothing" method="post">
<input type="hidden" name="-response" value="send.html">
(the response can be any other file)
Your Email Address:<br>
<input type="text" size=30 name="field1" value=""><p>
Subject: <br>
<input type="text" size=30 name="field2" value=""><p>
Message:<br>
<textarea name="field3" rows=4 cols=40 wrap=soft></textarea>
<p><input type="Submit" name="nothing" value="Send Email">
</form>
```

The response file "send.html" would contain the following inline:

```
[inline: emailhost="mail.domain.com", emailfrom="sender@domain.com",
emailformat="email.txt", emailto=form_param:"field1",
emailsubject=form_param:"field2", "field1"=form_param:"field1",
"field2"=form_param:"field2", "field3"=form_param:"field3",nothing]
[/inline]
```

The email can be sent since emailhost, emailfrom, and emailformat have hard-coded values. The other two required fields use the form parameters from the previous form submission.

Consequently, each field within the specified email format file needs to be set to the previous form parameter ("field1"=form_param:"field1", "field2"=form_param:"field2", "field3"=form_param:"field3"). When the email message is processed, a new action results and the form parameters are lost.

Conditional statements can operate with inlines to send email only if all required email tag values are provided.

Database Substitutions for Email Tags

Several of the email notification tags can take database field value substitutions rather than just hard-coded values. With this capability, it is possible to have an email message sent a person submitting the form or to some other address taken from a FileMaker Pro database field. Keep in mind that all fields must have a value for the message to be successfully sent, so make sure a value is added to each tag.

The tags that can take substitutions are: -emailfrom, -emailformat, and -emailsubject. These tags can be assigned field values from all records in the found set, and can appear several times in a single email message. It is also possible to provide a mixture of field substitution values and hard-coded values.

```
<input type=hidden name="-emailfrom" value="admin@domain.com">
<input type=hidden name="-emailto" value="field: FieldName">
```

If the -emailhost, -emailfrom, -emailformat, or -emailsubject tags are set to a field for database substitution, the field value is obtained from the first record in the found set. Each of these tags may appear only once in a single message.

If the fields are square-bracketed ([field: FieldName]), the field value substitution occurs when Lasso first processes the format file. If the fields are not bracketed (field: FieldName), the substitution occurs after the form is submitted.

If square brackets are used to surround the "field" tag, the value used will be the current value for that field in the database, when the form was initially processed by Lasso. The syntax in this case is as follows:

```
<input type=hidden name="-emailto" value="[field: FieldName]">
```

Without square brackets for the field reference, Lasso will substitute field values after the Lasso action occurs. For example, on an add action:

```
<input type=hidden name="-emailto" value="field: email address">
```

First, the record will be added. Then, Lasso will gather field values to be substituted from the newly added record (or the specified record on a search) and process the message.

Fields are also not bracketed when a value is entered into the same form that is the source of the value. Text needs to be specified if

the values aren't already present in the database or if the visitor needs an option to change the necessary values. For example:

```
<input type="hidden" name="-emailsubject" value="field:email_subject">
Subject:<input type=text size=40 name="email_subject"
value="Sent from Lasso">
```

An email action can be initiated when a search is completed, but the -recid tags will be required in order to identify the record on which to perform the field substitution. With an embedded URL, a search could be embedded as:

```
<a href="Lasso.acgi?-database=name&-layout=name&
-response=path-to-file/response.html&-recid=[recid_value]&
-emailhost=mail.senderdomain.com&-emailfrom=field:name-of-field&
-emailto=field:name-of-field&-emailsubject=Email%20Delivered%20
by%20 Lasso&-emailformat=PathtoFile/Email_Format.txt&-search">
Send Email</a>
```

Only one field substitution can be specified as the value for an email tag. You cannot specify a field and additional text (or several fields) for one email tag. For example, the following syntax will not result in a field substitution:

```
<input type="hidden" name="-emailsubject" value="text goes here, field:id">
```

The subject line will appear as "text goes here, field:id" instead of including values from the database.

Checking Email Addresses

Lasso will not send email if a database substitution field is blank. For example, if an email field is on an "add" form and the email tags are set to send a message to the person entering the form, Lasso will log an error and fail to send an email message if the user does not enter an email address.

If you wish to send an email message to visitors who are to provide an email address on a form, make the email address field a required field in the FileMaker Pro database and have an error page alert the user to include the email address. An incorrectly entered email address will result in bad mail messages. You can compensate for this using Lasso "string" commands to parse the submitted value and error check, or use a JavaScript routine.

There are situations when you may want to check the email address and perform a second action on the reply page. For example, if a message should be sent to an alternate address if no email is entered, use a conditional statement on a reply page as follows:

```
<input type=hidden name="-emailhost" value="mail.senderdomain.com">
<input type=hidden name="-emailfrom" value="sender@senderdomain.com">

[if: email==""]
<input type=hidden name="-emailto" value="administrator@receiverdomain.com">
<input type=hidden name="-emailsubject" value="Record added without Email
Address">
<input type=hidden name="-emailformat"
value="path_to_file/Email_Format1.txt">

[else]
<input type=hidden name="-emailto" value="[field: email]">
<input type=hidden name="-emailsubject" value="Email Delivered by Lasso">
<input type=hidden name="-emailformat"
value="path_to_file/Email_Format2.txt">
[/if]
```

With this construction, no email message will be sent if the address does not exist in the database.

Sending Email from a Format File: An Example

What follows is an example template that can be used for Lasso format files with embedded email commands. This template is included within FM Link. This requires several fields to be added to a database to accept email related values, in this case: "email_subject," "email_to_field," "email_message," "email_time," and "email_date."

```
<html>
<head>
<title>Lasso — Send Mail Template</title>
</head>
<body bgcolor=#ffffd9>
<center>

<table width=550><tr>
<td><b><font size="+2">Send Email</font></b></td>
<td align=right><font size="+2"><a href="javascript:history.go(-1)">
Go Back</a></font></td></tr></table>
<hr width=550>

<form action="go.lasso?-update&-database=YourDBName&
-layout=YourLayoutName&-response=insertpath/Detail.html&
-recid=[recid_value]" method="post">

<input type="hidden" name="email_date" value="[server_date]">
<input type="hidden" name="email_time" value="[server_time]">
```

```
<table width=550><tr>
   <td><input type="hidden" name="-emailhost"
      value="mail.senderdomain.com"></td>
<td></td>
</tr><tr>
   <td><input type="hidden" name="-emailfrom" value="sender@
   receiverdomain.com"></td><td>
<input type="hidden" name="-emailto" value="field:email_to_field">
<input type="hidden" name="-emailsubject" value="field:email_subject"></td>
</tr><tr>
   <td><input type="hidden" name="-emailformat"
   value="insertpath/Email_Message.txt"></td>
<td></td>
</tr>

<tr>
   <td>-emailto:</td><td>
   <input type=text size=40 name="email_to_field"

value="[field:email_to_field]"></td>
</tr><tr>
   <td>-emailsubject:</td><td>
   <input type=text size=40 name="email_subject" value="Subject: Sent from
   Lasso"></td>
</tr>
<tr>
   <td valign=top>Message:</td><td valign=top>
   <textarea name="email_message" rows=4 cols=60 wrap=soft></textarea>
</td>
</tr>
<tr>
   <td>FIELD1:</td><td>[field:"fieldname"]</td>
</tr><tr>
   <td>FIELD2:</td><td>[field:"fieldname2"]</td>
</tr><tr>
   <td>FIELD3:</td><td>[field:"fieldname3"]</td>
</tr><tr>
   <td>FIELD4:</td><td>[field:"fieldname4"]</td>
</tr><tr>
   <td>FIELD5:</td><td>[field:"fieldname5"]</td>
</tr><tr>
   <td valign=top>FIELD6:</td><td valign=top>[field:"fieldname6",break]
</td>
</tr>
```

```
</table>
<hr width=550>
<p><font size="+2">Select button to send email message:</font>
<input type="Submit" name="-update" value="Send">

</form>
[include:"insertpath/footer.txt"]
</center>
</body>
</html>
```

FM Link also contains an email format file template as follows:

```
Email sent from Lasso on [field:"email_date"],
at [field:"email_time"][If:field:"email_message"==""][else]

[field:"email_message"][/if][If:(field:"fieldname1"=="")|(field:"fieldname2"=="")|
(field:"fieldname3"="")|(field:"fieldname4"="")|(field:"fieldname5"="")|
(field:"fieldname6"="")][else]

_____________________________________________

The record has the following details: [If:field:"fieldname1"==""][else]
FIELD1:   [field:"fieldname1",raw][/if][If:field:"fieldname2"==""][else]
FIELD2:   [field:"fieldname2",raw][/if][If:field:"fieldname3"==""][else]
FIELD3:   [field:"fieldname3",raw][/if][If:field:"fieldname4"==""][else]
FIELD4:   [field:"fieldname4",raw][/if][If:field:"fieldname5"==""][else]
FIELD5:   [field:"fieldname5",raw][/if][If:field:"fieldname6"==""][else]
FIELD6:   [field:"fieldname6",raw][/if]
[/if]
```

This template uses a series of [if: …] conditional statements to remove field labels for fields that have no values. This produces a cleaner result for your email message. See CHAPTER 14: CONDITIONAL STATEMENTS for more details.

If a form simply needs to forward email, the form can be submitted using a -nothing action. Lasso will process the page and send the email specified by the email tags without interacting with any FileMaker Pro database.

Sending a Conditional Email Response: An Example

An [inline: …] tag can also be used to send a customized email message to a specific recipient depending upon the form data submitted. To use different email format files for different recipients, multiple [inline: …] tags are needed (one for each message to be sent). Each of the values can be hard-coded with literal text; use

field values returned from the previous record (i.e., "-emailsubject=field:insert-fieldname"), or values entered on the previous form but not entered into a database (i.e., "emailsubject=form_param:insert-fieldname").

If messages are to be set based upon variables, use the [if: …] tag along with the [set_var: …] tag. This allows variables to be used within an inline statement. For example:

```
[if: field:"group"=="sales"]
   [set_var: "efmt"="sales_mail.txt"]
   [set_var: "message"="Message from Sales"]
   [set_var: "efrom"="sales@domain.com"]
[else: if: field:"group"=="marketing"]
   [set_var: "efmt"="marketing_mail.txt"]
   [set_var: "message"="Message from Marketing"]
   [set_var: "efrom"="marketing@domain.com"]
[else]
<P>No Mail Sent
[/if]
[inline: emailhost="mail.domain.com", emailto=(field:"email",raw),
emailformat=var:"efmt", emailsubject=var:"message", emailfrom=var:"efrom",
nothing]
[/inline]
```

Note that the "raw" parameter is used to prevent the value from being HTML-encoded by default.

Chapter 13: Includes and Inlines

The [include: …] Tag

The [include: …] tag is used to embed a static HTML or text file into a Lasso format file. The tag is used in the following form:

```
[include: "relative-path-to-file/filename"]
```

When Lasso parses the file containing an [include: …] tag, the contents of the "included" file are merged with the main file at the inserted location. The included file can contain Lasso substitution tags. With pre-planning and intelligent use of the [include: …] tag, one can significantly ease multi-page Web site administration. For example, a single [include: …] file may set a global "footer" or "navigation bar" across all pages in a Web site.

The [inline: …] Tag

The [inline: …] tag provides the ability to process multiple Lasso actions within a single format file. In essence, the embedded [inline: …] functions as a "format file within a format file." The [inline: …] container tag can appear in any post-Lasso format file. The syntax is as follows:

```
[inline: database="db_name", layout="Layout_name",
   insert name="value", action]
      ...HTML and Lasso tags go here
[/inline]
```

A database name and layout name are required. An action is usually specified, although if left blank, the nothing action is used.

All other Lasso tag and HTML elements can also be used.

Nested Inlines

It is possible to pass values from a nesting [inline: …] to a nested [inline: …] as long as the reference to the passed value is contained within the inline statement in the nested inline's opening tag (first tag in the [inline: …]…[/inline] container tag pair). If the value or reference is placed anywhere else, it will not be available, and will not display.

For example, an "Employees" database has the fields "first name," "last name," and "EmployeeID"; and another database, "EmpSalary," has the fields "EmployeeID,"and "salary". The first [inline: …] appears as follows:

```
<!-- start inline #1 -->
  [inline: database="Employees", layout="Summary", first name="Horatio",
    search]
    [record]
    [field: "first name"], [field: "last name"]; [field: "EmployeeID"]
```

The second inline appears as follows:

```
<!-- start inline #2 -->
  [inline: database="EmpSalary", layout="salary",
  EmployeeID=field:"EmployeeID", search]
```

The value from inline #1 is used for "field: EmployeeID" in inline #2. The example continues as follows:

```
    [field: "salary"] <!-- value from inline #2 -->
    [field: "first name"] <!-- values from inline #1 are not available,
      so this will be blank -->
    [/inline] <!-- end inline #2 -->
    [field:"first name"] <!-- values from inline #1 are now accessible again -->
    [/record]
    [/inline] <!-- end inline #1 -->
```

Note that when using multiple inlines, or when nesting [inline: …] tags, more memory should be allocated to Lasso in order to store data for multiple results.

Executing a FileMaker Pro Script

The following syntax executes a FileMaker Pro script within an inline:

```
[inline: database="employees", doscript.post="TheScript", scripts]
…. [/inline]
```

The [inline_result] Tag

The [inline_result] tag is used within an [inline: …]container primarily to test the results while error checking.

The [inline_result] tag can also operate as a test within a conditional statement. [inline_result] represents a standard set of coded response values. When included within the [inline: …], [inline_result] is substituted with text to indicate the success of the

inline operation. The [inline_result] tag can take one parameter (the keyword "code") as follows:

```
[inline_result, code]
```

"Code" presents the numerical value of the result as opposed to text.

For example, if the [inline: …] operation was not successful because the proper database was not open, [inline_result] would display "errAENoSuchObject" while the tag [inline_result, code] would display "-1728."

What follows are the possible [inline_result] response values:

Numerical	Textual	Description
0	noErr	Operation was successful.
-609	connectionInvalid	FileMaker is not running.
-1712	errAETimeout	The Apple Event timed out before a reply was sent.
-1728	errAENoSuchObject	The object was not found. Make sure the database and layout exist.
-17005	errAEBadListItem	Operation involving a list item failed.
-800	errRequiredFieldMissing	Value missing for required field for Add.
-801	errRepeatingRelatedField	Adding repeating related fields is not supported.
-702	errTooMuchData	Too much data was given to the CGI.
-10011	errAEInTransaction	Another transaction is in progress.
-108	memFullErr	Not enough memory to complete operation.
N/A	invalidUsername	An invalid username was supplied for a protected database.
N/A	invalidPassword	An invalid password was supplied for a protected database.
N/A	noPermission	User does not have permission to complete this operation.
N/A	fieldRestriction	A field restriction (such as "exactsearch," "exactupdate," or "exactdelete") is in place.

The [post_inline: ...] Tag

The [post_inline: …] tag allows one to execute an inline operation after the processed page is returned to the user.

This tag is comprised of one required attribute ("post_response"), several optional attributes, and any number of parameters. The syntax is as follows:

```
[post_inline: post_response="YourFileName.html", attribute="value",
name="value"]
```

The required "post_response" attribute must specify the path to the file Lasso will process.

The following attributes are optional:

- set_hours — Specifies the number of hours to wait before executing the specified operation.
- set_minutes — Specifies the number of minutes to wait before executing the specified operation.
- set_day — Specifies the day to execute the operation (Mon, Tue, Wed, Thu, Fri, Sat, Sun).
- set_month — Specifies the month to execute the operation (1 through 12).
- set_time — Specifies the hour of the day to execute the operation (1 through 24).
- set_week — Specifies the week of the month to execute the operation (1 through 4).

Values set in an inline are passed to the post-response page. The [form_param: …] tag can be used to display them.

An inline process can be carried out at the interval or time specified in the post inline tag. If no time is indicated, the action will occur as soon as Lasso is idle.

A post-inline can be used on a page that is part of a set of format files. This is useful if you want some action to occur but you do not want it to hold up the delivery of the response file to the user. In this case the [post_inline: …] tag defers the activity until the current action is completed.

The [post_inline: …] tag can execute in a file by itself. For example, to send an email each night at midnight, create a new format file named "midnight.lasso" which contains the following code:

```
[post_inline: "/path_to_file/send_message.lasso",set_time=24]
```

Save it to the "Lasso Startup Items" folder. Create another file named "send_message.lasso" which contains the five required email tags.

Chapter 14: Conditional Statements

The [if: ...] Tag

Conditional statements can add flexibility to your Lasso integrated database project by customizing Web pages according to current values or conditions. The Lasso container tags [if: …] [else] and [/if] are referred to as the conditional statement tags, and control HTML output on post-Lasso format files (files processed and returned by Lasso). They control the returned Web page by displaying specified text, HTML, or Lasso tags if a condition holds true. In addition, the [while] tag can be used to repeatedly display specified text or re-evaluate the expression as long as the resulting condition is valid.

When a conditional statement is evaluated, it has either a true or false result. When a Lasso action is submitted, the conditional tags direct Lasso to check a comparison expression, and if the condition is "true" the text (or tags) between [if: …] and [else], or [if: …] and [/if], are returned. If the condition evaluates to false, then the text (or tags) which follows the [else] or [else: if: …] are referenced, and the remainder of the page after the ending [/if] tag is processed. If neither [else] tags were used, then nothing appears when the first condition is false (but the remainder of the page is processed).

An example:

```
[if: field:"cards"=="21"]
    Blackjack!
[else]
    Deal Again
[/if]
```

If the returned field value is equal to 21 then "Blackjack!" is displayed. If not, then the result is "Deal Again."

Any Lasso substitution tag can be used within the [if: …] expressions, not just literal text. There are several Lasso tags that enhance the power of conditional statements. For example, the number of repetitions within several types of repeating tags can be tested and HTML elements or Lasso generated values returned in a specified

pattern. This is especially useful if you want to put records in rows of a table.

Another example of the usefulness of the conditional tag is the ability to specify the content of an error response page based on the previous action of the user. To do so, check the last action taken using the [lasso_action] tag and have the page processed based on what that action was. Also, the [nfound] tag can be used on a hitlist page to return the details of a single record when a single record is found. The condition would need to surround the entire page with an [include: …] tag for the detail page between [if: nfound=="1"] and [else] and the usual hitlist page elements from <HTML> to </HTML> located between the [else] and [/if] tags. The detail page is returned if one record is returned, and the hitlist page if there are no more. There are many more possibilities beyond these few examples.

Note: Lasso will report whether a conditional statement is badly formed within the reply file itself. The error message will be in red and describe what was wrong with the syntax used.

Components of the [if: …] Tag

The [if: …] tag encloses a conditional expression that is made up of one or more conditional statements and evaluated as a whole. Each conditional statement consists of two comparison fields that appear on either side of an operator. Multiple comparisons can be concatenated together within one conditional by using "and" and "or" logical operators within the expression. In other words, the expression can test a series of conditions as a whole, and result in "true" if either all expressions are true (and), or one of the expressions is true (or). Also, conditional [if: …]…[else]…[/if] statements can be nested to evaluate a series of conditions.

There are three elements that make up a conditional expression:

1. The comparison fields that appear on either side of a comparison pair.
2. The operator that determines how to compare those values.
3. An optional concatenation symbol ("and" (&) or "or" (|)) which is used to create multiple condition expressions.

Each element used to construct the [if: …]…[else]…[/if] tag is explained below. The general form of the tag is:

```
[if: (comparison-value) operator (comparison-value)]
   ...HTML if condition is true...
```

```
[else]
    …HTML if condition is false…
[/if]
```

Note: The [/if] tag cannot be referenced as [endif] as was possible in previous versions of Lasso.

Comparison Values

The "comparison value" used within the [if: …] expressions can either be a hard-coded literal value, a value returned from a FileMaker Pro field, or a Lasso substitution tag. Comparison values are not case-sensitive and can include spaces within text values.

Literal Value

The literal value is a specific text or number to be used for the comparison. Here's an example of a hard-coded literal value used in a comparison:

```
[if: field:"Employee Number" <= "10"]
    <strong>Wow! This guy must have founder's stock!</strong>
[else]
    <font size="-1">Oh, well. Missed the boat.</font>
[/if]
```

String or literal values should use quotes to designate the start and end of the comparison value. Also, if specifying blank or nothing, use two quotes with no space (""). It is highly recommended that quotes be used to denote the comparison value, but it is not required. Quotes allow Lasso to evaluate a comparison more quickly since they denote that the text is a value or string literal and not a Lasso tag (Lasso checks unquoted text against a list to see if it is a tag). If specifying a literal value without quotes, Lasso treats everything from the end of the operator to the closing right square bracket (minus leading and trailing spaces) as the comparison value. To use an actual quote in the string literal, use a backslash as an escape. For example, the following will check if the value is "I said "something"":

```
[If: field:"quote"=="I said \"something\""]
```

Note: The use of quotes surrounding literal values was introduced with Lasso 2.0.

Field Values

Field values can be substituted into the conditional statement. If a field name is specified, this field must be one that is located on the layout (and database) specified on the previous format file or

embedded URL that invoked the file where the conditional tags are located. To specify a field, use the "field" tag, followed by a colon, and then the name of the field in quotes, as follows:

```
[if: field:"Salary" > field:"Expenses"]
```

The following is also correct:

```
[if: field:"Salary"> field:"Expenses"]
```

In order for a field value to be substituted into the conditional statement, the field name must be preceded with "field:" and then the name of the field. This is true for fields on either side of the "name=value" pair. Don't use brackets ("[ ]") to enclose the field tag and field name. Using quotes to enclose the literal name of the field is optional, however, it allows Lasso to more quickly locate the field.

Here's an example showing a comparison between two field values (Assets and Liabilities):

```
[if: field:"Assets"> field: "Liabilities"]
<font color="green"> Nice going, dude. You're solvent.</font>
[else] <font color="red">Time to cut up those credit cards.</font>
[/if]
```

A conditional statement can be useful if you want to check if a guest has left an empty field. For example:

```
[if: field:"fieldName"==""]
      The field is blank
[else]
      Do something here
[/if]
```

The field value is retrieved as a result of the Lasso action that called up the page where your conditional tags are placed. Thus, in order for a field value to be evaluated, a record in a specific database must have been located by Lasso.

Other Lasso Tags

Lasso substitution tags can be used since they are substituted with a value when returned. For example, [nfound], or [client_id] can be used. Lasso tags are never surrounded by quotes. For example:

```
[if: token_value=="test"]
      ….do this
[else]
      ….do that
[/if]
```

Or:

```
[if: nfound=="1"]
    [if: field:"identifier"=="john"]
        …display detail 2 of form
        (can use [include: "path/detail_john.html"])
    [else]
        …display detail 1 of form
        (can use [include: "path/detail.html"])
    [/if]
[else]
    …display hitlist form
[/if]
```

Or:

```
[if: cookie:"userid" == field:"guest"]
     …update form…
[else]
    …add form…
[/if]
```

Using conditional statements with sub-container tags is especially useful for determining specific results within the associated container tag. Each conditional statement applies only to the tag it is contained within. For example, within [repeating:-…]…[/repeating] tags the comparison to the "repeat value" applies only to the field specified by the [repeating: …] tag, i.e.:

```
[repeating: "Field1"]
   [if: repeat_value=="yes"]
      This HTML will appear when the
      value of Field1 is "yes".
   [/if]
[/repeating]
```

When the [if: …] tag is used within portals, fields within the portal must be specified with the full relationship name and field name, and be separated by colons as follows:

```
[if: field:"relationshipname::fieldname"=="value"]
[else]
    <!-- Do something here -->
[/if]
```

Operators

The operator determines how the comparison is made. The operators supported by Lasso include:

Operator	Meaning
==	equal to
!=	not equal to
>	greater than
>=	greater than or equal to
<	less than
<=	less than or equal to
>>	contains

Note that a double equals symbol ("==") must be used for the "equal to" operator. Single equals signs should not be used as was the case with previous versions of Lasso.

Some examples:

```
[if:string1 >> string2 ]
```

is true if string1 contains string2.

```
[if: "field:company_name" >> "world"]
```

would be true for:

```
Blueworld
World webs inc
huge worldnet Co.
```

and false for:

```
test inc.
blueplanet
```

The following example shows how to apply special HTML formatting to select values.

```
[if:field:"title" == "president"]
    <strong>[field:"title"] [field:"name"] </strong>
[else]
    [field:"title"] [field:"name"]
[/if]
```

would return a result similar to:

```
Manager Tom Wheating
President Bill Doerrfeld
Vice-President Al Gore
Retired Bob Dole
```

Concatenation Symbols for Multiple Comparisons

Multiple comparisons can be achieved within a comparison expression. Logical operators are used to concatenate several conditions together so each condition is evaluated separately and then compared between one another. Multiple conditional statements are concatenated together using double ampersand characters "&&" for "and" and double pipe characters "||" for "or." Comparisons are evaluated first (<, >, =, etc.) and then the concatenated comparison is evaluated as a whole. Parentheses can be used to separate the order of the comparisons, with the most-nested parentheses evaluated first.

Using the "or" logical operator, the entire expression evaluates to "true" when any one of the comparisons is true. For example, the following expression would be "true":

```
[if: true || false]
    ...display this
[/if]
```

Using the "and" logical operator, the entire expression evaluates to "true" when all comparisons are true. For example, the following expressions are both "true":

```
[if: true && true]
    ...display this
[/if]
[if: true && (true || false)]
    ...display this
[/if]
```

"True" and "false" simply represent the result of a condition.

Here is how a more complex expression would use concatenated symbols:

```
[if: value1 < value2 && ( value3 == value4 || value5 > value1)]
    ...display this
[/if]
```

Here is another example:

```
[if: (client_ip=="196.46.198.222" && server_day == "Sat") |
client_ip=="259.204.68.2"]
    It's Kyle!
[else]
    It's not Kyle!
[/if]
```

Within this expression, the first IP address and day of the week are compared. The comparison is true if either the result of that is true, or the second IP address comparison is true. Without the parentheses, the result would not be the same since the first IP address would result in a true result, no matter what day it was.

Using multiple conditional statements avoids using nested [if: …] statements. For example, the following would check to see if someone was accessing your site on the weekend (on Saturday or Sunday):

```
[if: (server_day == "Sat") | (server_day == "Sun")]
        weekend text
[/if]
```

The [while: …] Tag

The [while: …] tag is a looping tag that repeats as long as a certain condition is true. It is a cross between the [if: …] and [loop: …] tags. The conditional statement follows the same specifications used with the [if: …] tag. The basic form of the tag is:

```
[while: (comparison-value) operator (comparison-value)]
   ...HTML if condition is true...
[/while]
```

Whatever is found between the [while: …]…[/while] container tag will be repeated as long as the condition evaluates to "true."

The [loop_count] tag is a useful sub-container tag to use with the [while: …] tag. It allows the current loop value to be compared so the [while: …] operation will repeat a specified number of times.

```
<p>Loop while loop_count <="20"<br>
[while : loop_count <="20"]
   [loop_count],
[/while]
```

Conditional Statements Based on Repetition

Conditional statements can test for the number of repetitions within a [record], [repeating: …] or [portal: …] container. The result of this test can then be used to apply specific HTML formatting or modify specific Lasso-returned values. For example:

```
[record]
      ...blah...
      [if: repetition=="2"]
```

```
            ...blah blah...
        [/if]
[/record]
```

The above will insert "…blah blah…" for every other record. If the test was repetition== "3" then "…blah blah…" would be printed for every third record, and so on.

When either of the [record], [repeating: …] or [portal: …] tags are nested within one another, the tag that the repeating value applies to is in the following order of precedence: [repeating: …], [portal: …], [record].

The [repetition: …] tag is useful when it is necessary to end the rows of a table at a specified point. Using [repetition], it is possible to put records in rows of a table and to change the look of every other row as follows:

```
<table>
   <tr><td>Name</td>
<td>Rank</td>
<td>Serial Number</td></tr>
[record]
   <tr><td>[field:"name"]</td>
<td>[field:"rank"]</td>
   <td>[field:"serial"]</td></tr>
[if: repetition=="2"]
   <tr bgcolor="#dddddd">
<td>[field:"name"]</td>
   <td>[field:"rank"]</td>
<td>[field:"serial"]</td></tr>
[/if]
[/record]
</table>
```

The following puts each repeating value in a cell of a table going across the row and starts a new table row after every fifth repetition:

```
<table><tr>
[value_list:"typeclient"]
<td><input type=radio name="typeclient" value="[list_value]" [checked]>
[list_value]</td>
   [if: repetition=="5"]</tr><tr>
   [/if]
[/value_list]
</tr></table>
```

The following syntax shows how to put each value of a radio button selection list into its own cell and begin a new row of the table at every fifth repetition:

```
<table><tr>
[value_list:"choice1"]<td>
<input type=radio name="choice1" value="[list_value]" [checked]>
[list_value]</td>
[if: repetition=="5"]</tr>
<tr>[/if]
[/value_list]
</tr>
</table>
```

In the above example, each radio button selection will be separated into its own cell, the row will end after 5 checkboxes, and start again in a new row. The same technique can apply to checkbox selection lists as follows:

```
<table><tr>
[value_list:"qualifications"]
<td>
<input type="checkbox" name="qualifications" value="[list_value]"[checked]>
[list_value]</td>[if: repetition=="5"]
</tr>
<tr>
[/if][/value_list]</tr>
</table>
<input type="hidden" name="qualifications" value="">Unselect All
```

In the above example, each checkbox will be separated into its own cell, the row will end after five checkboxes, and start again in a new row.

Comparisons to Value Lists

When using conditional tags within [value_list: …] tags, additional options are available. The [list_value] tag can be used on either side of the comparison operator to substitute the items from the current value list. Furthermore, the HTML attribute "checked" (which is used with radio buttons or checkboxes) or "selected" (used with pop-up lists or scrolling lists) can be used to determine if the list value is selected in the database. The Lasso tag [checked] will return the text "checked" whenever a list value item has been selected for the current record.

When creating a comparison for a field formatted with a value list, the comparison is made to the value list item rather than to the

field itself. In other words, one constructs the [if: …] tag within the [value_list: …]…[/value_list] container and makes the comparison to the [list_value]. Comparisons can be made only to the field specified by the [value_list: …] tag. For example:

```
[value_list:"temperature"]
   [if: list_value == "123"]
      --- This is the item 123 ---
   [/if]
[/value_list]
```

Note that in the following, the Lasso tag [checked] is not quoted, but the text "checked" is:

Select a shift:

```
[value_list: "shift"]
<tr><td>
      <input type=radio name="shift" value="[list_value]" [checked]>
</td><td>
      [if: list_value=="1"]
            This is shift 1.
      [/if]
      [if: list_value=="2"]
            This is shift 2.
      [/if]
      [if: list_value=="3"]
            This is shift 3.
      [/if]
</td><td>
      [if: checked == ""]
            This is the currently assigned shift.
[else]
[/if]
</td></tr>
[/value_list]
```

The [selected] tag achieves the same results, as follows:

Select a group:

```
   <select name="group">
            [value_list:"group"]
            <option value="[list_value]"[if: selected ==""]
selected[/if]>[list_value]
            [/value_list]
   </select>
```

However, in this case the syntax may be specified as follows:

```
<select name="group">
    [value_list:"group"]
    <option value="[list_value]"[selected]>[list_value]
    [/value_list]
</select>
```

When dynamically returning a value list from the database, it is not possible to specify that the list value is different than what is displayed, for example:

```
<option value"one-thing">value item returned by Lasso from database
```

When Lasso gathers the listed items, the value is always exactly how the list is defined in the FileMaker Pro database. However, using conditional statements you could compare the list value, and based on that change the list items that are displayed:

```
<select name="field1">
    [value_list:"field1"]
        <option value="[list_value]">[If: [list_value]=="some item on list"]
        Display something else [else] [list_value][/if]
    [/value_list]
</select>
```

To check multiple list items, you need to check each item on the list. For example:

```
<select name="field2">
  [value_list:"field2"]
    <option value="[list_value]">
      [If: list_value=="1st item on list"] Display Something other than first
      [else: If: list_value=="2nd item on list"] Display Something other than
second
        [else: If: list_value=="3rd item on list"] Display Something other than
third
        [else: If: list_value=="4th item on list"] Display Something other than
fourth
    [else] [list_value]
    [/if]
  [/value_list]
</select>
```

The use of conditional statements within the [value_list: …] tag is just one example of using conditional statements within container tags, other examples can be found in other chapters of this manual.

Using an [inline: …] with Conditional Statements

The following is an example use of how to combine the [inline: …] tag with conditional statements. Place this code in any file processed by Lasso (a post-Lasso form, or a suffix-mapped file that is set in the Web server to use Lasso), to get a table listing all the employees whose first names begin with "Erik." This can even be placed on a hitlist page that is searching a different database.

```
[inline: database="Employees", layout="Summary", "First Name"="Erik",
search]
current action is: [lasso_action]
<table>
[record]
<tr>
      <td>[field: "First Name"]</td>
      <td>[field: "Last Name"]</td>
</tr>
[/record]
</table>
[if: inline_result:code =="0"]
<br>searched with no problems<br>
[else]
problem connecting to DB: [inline_result] [inline_result,code]
[/if]
[/inline]
```

The following is another example which shows how to use an [inline: …] to send an email whenever someone from outside your domain views a page:

```
<html>
<head>
<title>Text</title>
</head>
<body>
[if: client_ip != "202.230.55.*"]
      [inline: emailhost="domain.com",emailfrom="zero@domain.com",
      emailto="john@domain.com",emailsubject="This is the subject",
      emailformat="/Stranger.html", nothing]
      An email was sent, notifying us of your presence.<p>
      [/inline]
[/if]
</body>
</html>
```

Note that specifying the -nothing action is not necessary.

Chapter 15: Logging Lasso Activity

There are several ways to keep track of activities resulting from Lasso actions. It is possible to record details of this activity into a log, or to display the activity in the active log window. Log information can be saved to either a target text file or specified fields within a database (when combined with an inline routine).

In addition, Lasso also automatically creates an error log file. A Lasso error log is created automatically with all editions of Lasso (CGI, Plug-in and Server). The file is called "Lasso Error.log" by default. If deleted, or renamed, a new one will be created. The error log file is always saved at the root level of the Web serving folder. This location cannot be changed. Lasso will record the numeric error code and any associated error text, as well as the date and time that the error occurred. There is no way to customize the Lasso error log.

The [log: …] Container Tag

The [log: …] tag allows for information relating to the current Lasso action to be written to a text file or displayed in the server (or Lasso.acgi) log window. Logging activity occurs when Lasso has idle time. There are two parameters available: the actual path to a text file, and "window."

Path Parameter

If a path is indicated, the values specified between the [log: …]…[/log] container are appended to the end of the targeted file. It is not possible to overwrite the contents of this file. The file path must be indicated from the root level of the Web serving folder.

For example:

```
[log: "/path/to/logfile"] the log text [/log]
```

Several separate logs can track different activities with each writing to a different file or all writing to the same file. Simply add additional log tags to a format file. Do not nest these tags.

If a path and file is not specified, a default log file will be created at the root level of the server and titled "Lasso.log."

Window Keyword

The [log: …] tag can take the "window" keyword as a parameter instead of a file and path specification. "Window" instructs Lasso to output the text or tags between the opening and closing [log: …] tags to the Lasso log window instead of a file.

For example:

```
[log: window] [server_date,short],[server_time,short],[cookie:id]
[/log]
```

For the Lasso Plug-in, the text is displayed within the server log window. This information is useful for troubleshooting.

Content of Log

Lasso will log whatever is between the opening and closing [log: …] tags. For example, tabs or commas can be used to create a delimited file. Any Lasso substitution tag can be used within the log tags, including fields from the specified database and layout. If each log entry is to appear on its own line in the log file, use a line break at the end of the line as follows:

```
[log]the log text with a line break at the end
[/log]
```

Any line breaks in the HTML are put in the log file and are not ignored as is common with HTML, thus:

```
[log]test[/log]
```

is:

```
testtesttesttesttesttesttesttesttesttest
```

And:

```
[log]test
[/log]
```

is:

```
test
test
test
```

One example is to use the [log: …] tag on a no-results error file to report the activity that led to no records being found. The following syntax would include the date, time, name of the database, what field was searched on and how, and the name of the visitor (if Lasso Security is employed to log someone into the site):

```
[log: "/logs/noresult.log"]no result:[server_date,short],
   [server_time,short],[database_name], [search_args][loop_count]:
   [search_field] [search_op] [search_value], [/search_args],[client_username]
[/log]
```

Note that there should be a line break after the last item to be logged so that each log event will appear on a separate line. The file created by this example will appear as:

```
no result:5/16/97,12:25 PM,Employees,1) job equals carpet,TomR
no result:5/18/97,11:05 PM,Employees,1) name equals Jesus, 2) job equals
   haberdasher,John
no result:5/18/97,11:15 PM,Employees,1) name begins with snellergish,Pete
no result:5/22/97,00:11 PM,Employees,1) Job begins with snake,TomR
```

Multiple log tags can be used to have Lasso log to both a text file and to the log window.

Activity Log Using [inline: …]

To log activity to a database, use the [inline: …] tag. The inline activity occurs as the page is processed and this secondary and separate action can be hidden from the actual purpose of the returned page. Using the [if: …] conditional tag, you can set the [inline: …] to add values to the log database whenever a certain condition is met. In effect, specialized logs could be created to track specific activities with log information stored in various targeted databases. For example, you could log whenever someone performs a search as follows:

```
[inline: database="EmpLog", layout="List","User IP"=client_ip,"Num
Hits"=nFound,"First Name Term"=form_param:"First Name","Last Name
Term"=form_param: "Last Name", "Group Term"=form_param:"Group", "Shift
Term"=form_param:"Shift","TheDate"=server_date,"TheTime"=server_time,
add] [/inline]
```

This will add the IP address of the searcher, the number of hits found, search criteria, and a date/time stamp. The search criteria is obtained using the [form_param] tag which allows for the previously entered values to be captured. When the page is processed the fields that are contained in the inline tag are added to the database. In the above example, the "EmpLog" database would contain records with field data as follows:

User IP: 102.34.56.21
Num Hits: 15
First Name Term: John
Last Name Term: Doorey
Group Term: fixtures
Shift Term: night
TheDate: 5/5/97
TheTime: 21:23:31

Each entry would appear as a new record in the database (the field names are on the left, sample values on the right).

Since there is no text between the [inline: …]…[/inline] container, nothing displays on the returned page. Text could be placed there to indicate that the logging occurred or to display values from the specified database and layout.

Chapter 16: HTTP Content and Controls

The [referrer] Tag

Lasso provides a link to the document's referrer using the [referrer] tag. The referrer is passed to Lasso from the browser and is the URL of the page the user visited just prior to the currently viewed page. The referrer can serve as a "back" link in a format file, but only when available. Here is the syntax:

```
[referrer]Insert Link Text Here[/referrer]
```

When processed, Lasso automatically substitutes a referrer link in the format:

```
<a href="referringpage.html">Insert Link Text Here</a>
```

Since the referrer uses the URL of the previous page, it cannot be used to return to pages created by a "post" action. However, it will work if the previous page was generated by Lasso actions and commands.

Time/Date Stamping

- **[server_day] —** Displays the day of the week. Optional parameters are "short" and "long"; the default is "short." The short form is a three-letter abbreviation in all capital letters (MON, TUE, etc.). The long form is in mixed case (Monday, Tuesday, etc.).
- **[server_date] —** Displays the current date. This tag takes one of three optional parameters (the default is "short"):

short:	1/1/97 (month/day/year)
abbrev:	Wed, Jan 1, 1997
long:	Wednesday, January 1, 1997

- **[server_time] —** Displays the current time. This tag takes optional parameters "short" and "long"; the default is "short." Long includes seconds, short does not.

Client Content

Several substitution tags take advantage of information gathered from a visitor's browser. These may be used to simply display information about visitors, or used in conjunction with conditional expressions to dynamically route visitors (see CHAPTER 14: CONDITIONAL STATEMENTS). Browser Content tags include the following:

- **[client_addr]** — Displays the client's domain name if domain name lookups are enabled in the Web server. Displays the client's IP address if domain name lookups are disabled.
- **[client_ip]** — Displays the client's IP address.
- **[client_type]** — Displays the client's browser type. [client_type] is also known as the "user agent" and is a somewhat arbitrary text string of up to 256 characters sent by the browser in the HTTP request header. In theory, client type can be used to determine what sort of browser the person accessing your Web site is using.
- **[client_username]** — Displays the username (from HTTP basic authentication).
- **[client_password]** — Displays the password (from HTTP basic authentication).

Client Domain Within [if: …] Tag

The domain name of the user (as presented by the browser in the incoming HTTP request) can be used within [if: …] tags in the following manner:

- Only the equals operator ("==") can be used to compare domain names.
- Only hard-coded literal values can be used, not fields.
- Domain names on the right side of the equals operator may take an asterisk as a wildcard character for the leading portions of the domain name.
- There must not be any blank spaces in domain names on the right side of the equals operator.
- Multiple domain names may appear on the right side of the equals operator, separated by commas.

Some examples:

```
[if: client_addr=="harry.blueworld.com"]
      <b>Hi, Harry!</b>
[/if]
[if: client_addr=="*.blueworld.com"]
      <b>Welcome, Blue Person!</b>
[/if]
[if: client_addr=="*.edu"]
      <b>Be sure to ask about educational discounts!</b>
[/if]
[if: client_addr=="santa.claus.com, easter.bunny.com"]
      <b>Get real!</b>
[/if]
[if: client_addr=="*.everyware.com, *.macweb.com"]
      <b>Our wives are prettier, our kids are smarter, our plug-in is faster.</b>
[/if]
```

If DNS lookups are disabled in the Web server, the [client_addr] as reported by the browser is a numeric IP address and not a domain name. Therefore, all conditional statements using [client_addr] as a domain name (rather then the IP address) within the [if: …] tags will fail (will evaluate to false).

Client IP Address Within [if: …] Tag

The IP address of the user (as presented by the browser in the incoming HTTP request) can be used within [if: …] tags in the following manner:

- Only the equals operator ("==") can be used to compare IP addresses.
- Only hard-coded literal values can be used, not fields.
- IP addresses on the right side of the equals operator may take an asterisk as a wildcard character for trailing portions of the address.
- IP addresses on the right side of the equals operator must not contain any blank spaces.
- Multiple IP addresses may appear on the right side of the equals operator, separated by commas.

Some examples:

```
[if: client_ip==168.191.97.167]
      <b>Hi, Harry!</b>
[/if]
```

```
[if: client_ip==168.191.*]
    <b>Welcome, Person Whose IP Address Starts with 168.191!</b>
[/if]

[if: client_ip=="207.107.95.*", "206.24.108.*"]
    <b>Prettier, smarter, faster.</b>
[/if]
```

Header

An HTTP header can be produced by Lasso using the [header]…[/header] container in a response file. Any text found within the [header]…[/header] container will become the actual HTTP header for the response file. Several examples using the [header]…[/header] container are included below.

The rules governing the use of the [header]…[/header] container are as follows:

- The [header]…[/header] container may appear anywhere in a Lasso format file.
- Any Lasso tags may be used within the [header]…[/header] container.
- The literal string "HTTP" (all uppercase, no quotes) must appear somewhere within the [header]…[/header] container. All text preceding the HTTP string will be removed from the header.
- Lasso tags that modify the HTTP header (currently [set_cookie: …] and [content_type: …]) must appear within or after the [header]…[/header] container in the format file.
- All carriage returns (CR) within the [header]…[/header] container will be replaced with a CR/LF (carriage return/line feed) pair. As HTTP headers are always terminated by two carriage return/line feed (CR/LF) pairs, Lasso will not allow consecutive CR/LFs anywhere within the [header]…[/header] container. However, two CR/LF pairs are always required before the ending [/header] tag. Lasso always insures that there are exactly two CR/LF pairs to mark the end of the header.
- No spaces can be used at the start of the header line.

Header Construction

A HTTP header must follow the Hypertext Transfer Protocol (HTTP) standard maintained by the World Wide Web Consortium

(W3C). The WC3 guidelines for object headers specifies that several lines must be included. The first line being the status line:

```
<http_version> <status_code> <reason_line>
```

The numeric values for various status codes to HTTP requests can be found at:

```
http://www.w3.org/Protocols/rfc2068/rfc2068
```

For example, to redirect an HTTP request, the status code "302" is used as follows:

```
HTTP/1.0 302 FOUND
```

The next few lines must contain one or more HTTP directives. For example, an HTTP redirect request is specified as follows:

```
HTTP/1.0 302 FOUND<CrLf>
URI: Location: http://www.domain.com/path-to-file/Add_Reply.html<CrLf>
Server: Lasso/2.5
<CrLf>
<CrLf>
```

The symbols "<CrLf>" indicate the presence of a carriage return and line feed. The redirect URL can be specified by either "URI" (Uniform Resource Identifier) or "Location," or both, to provide greater compatibility with a variety of browsers (having both is optional). The "Server" line is optional, although other HTTP requests may require it.

Submit Without Reloading or Redirecting the Page

Using the [header]…[/header] container, a form can be submitted to a database without changes on the browser side. The form will remain as is. No reloading or redirecting will occur. the response page should include the following:

```
[header]
HTTP/1.0 204 FOUND
Server: Lasso/2.5
<CrLf>
<CrLf>
[/header]
```

This may be useful if you have an update form for a long list of records and you want to update each without searching for the list all over again. Of course, the contents of the updated records will not change until the record is found again.

Counting the Number of Times a Link is Selected

The following example extends the functionality of the "Rotating Banners" example included in CHAPTER 9: GRAPHICS AND MULTIMEDIA. This example allows for images to be randomly displayed. Additionally, when an image is selected, the user is taken to the URL associated with that image. The number of times an image link is selected is also recorded.

In the example, "FieldName1" is the field that stores the banner graphics. "FieldName2" is a field that stores the URL for an image.

The first action of selecting a link, actually submits a form using a "nothing" action. In other words, the image is used as a submit button as follows:

```
[inline: database="YourDBName", layout="YourLayoutName", random]

<form action="action.lasso?-response=process.lasso" method="post">

<input type="hidden" value="[field:"FieldName2",raw]" name="-token">
<input type="hidden" value="FieldName3" name="[recid_value]">

<input type="Image" src="[field:"FieldName1",raw]" name="-nothing">

</form>
[/inline]
```

The processed file "process.lasso" contains an inline to record that the link was selected. The header tag is then used to redirect as follows:

```
[inline: update, database=YourDBName, layout=YourLayoutName,
visitcount=(math-add:field:"visitcount",1),
recid=form_param:"FieldName3"][/inline]
[header]
HTTP/1.0 302 FOUND
Location: http://[token_value]
URI: http://[token_value]
Server: Lasso/2.5
<CrLf>
<CrLf>
[/header]
```

Substitute "YourDBName" and "YourLayoutName" for actual values. "visitcount" is the field that collects the data on how many times the image was linked to. Note that the record ID is passed using a form parameter (form_param:"FieldName3").

Timeout

The -timeout tag sets the time that Lasso will wait for a response from FileMaker Pro. The value is measured in seconds and can be a minimum of 10 and a maximum of 300 seconds (5 minutes). This parameter is optional. If the tag is not used, the default is 60 seconds.

The timeout setting controls only the time Lasso will wait for a response from FileMaker Pro. The Web server also will have a timeout setting that applies to Lasso or any other CGI. For example, for WebSTAR, the default CGI timeout is one minute. The Web server's CGI timeout may need to be increased if Lasso's timeout is increased. The -timeout tag is specified as follows:

```
<input type="hidden" name="-timeout" value=120>
```

In the example above, the timeout value is set to 120 seconds.

Content Type

The [content_type] tag allows the user to override the default content type of "text/html." The [content_type] tag must be the first entry in the format file, otherwise it is ignored by Lasso. For instance, if you wanted to have the document returned from Lasso be processed by the client's browser as an SGML document rather than a standard HTML document, specifying the following:

```
[content_type: text/sgml]
```

The [content_type] tag is ignored if it appears within a format file that is included in another format file via the [include] tag.

PIXO Support

PIXO ("Plug-In Cross-Over") is a "standardized" way for Web server plug-ins to communicate and cooperate for better performance and advanced integration.

PIXO allows plug-ins loaded into the same server to call each other without the overhead of Apple Events or TCP/IP.

Lasso supports the PIXO standard by allowing other Web server plug-ins to send text to Lasso for processing. Lasso currently cannot send text to other plug-ins. The Lasso Plug-in's PIXO service name is "Lasso_Plug-in_PIXO."

Chapter 17: Retrieving Values

The [form_param: …] Tag

The [form_param: …] tag allows values entered into a form to be retrieved on the response file. These values can either be displayed or inserted into HTML. For example, the [form_param: …] tag can be used to auto-populate "add," "search," or "update" forms or inline operations, without the values being saved into a database.

The form parameter value is retrieved by specifying the name of the field the value was associated with in the previous Lasso action. The source of the value is either a field in a form or the value that was paired to a field in an embedded URL or inline action. The format is as follows:

```
[form_param: "FieldName"]
```

The value "FieldName" is any "field" used in the previous format file. It can be the actual name of a field, or any made-up name if the data is not submitted to a database.

The [form_param: …] tag can be used in any response form, but is most useful in a "No Results" page. In this case, including the previous search parameters allows the user to modify, as opposed to re-type, the search parameters.

Examples of how to format the various types of field inputs are shown below.

Use of the [form_param: …] Tag

To use a form parameter value from the data entered into a field called "userid," use one of the following on the reply page.

Within a form:

```
<input type="text" size=30 name="name" value="[form_param:"userid"]">
```

As an embedded URL:

```
<a href="action.lasso?-search&-database=employees&
-layout=main&-response=detail.html&-op=eq&name=[form_param:"userid"]">
Search </a>
```

In an inline:

```
[inline: search, database="employees", layout="main", operator="eq",
"name"=form_param:"userid"]
   name = [field:"name"]
[/inline]
```

The [form_param: …] tag can be used to populate values in a pop-up selection list as follows:

```
FIELD: <select name="fieldname3" size=1>
   <option value="" [If:form_param:"fieldname3"==""]selected>
—- Select Here —-[else]>[/if]
[If:form_param:"fieldname3"==""][else]
   <option value="[form_param:"fieldname3"]" selected>
[form_param:"fieldname3"][/if]
   <option>pop-up1
   <option>pop-up2
   <option>pop-up3
   <option>pop-up4
</select>
```

An [if: …] conditional statement can be utilized to eliminate unwanted repetitions of the values in the selection list.

Count Keyword and Number Parameter

When multiple occurrences of a field (repeating fields, etc.) are submitted, the [form_param: …] tag will return all fields' values concatenated together.

In such cases, the "count" keyword can be used to return the number of values the specified [form_param: …] has. Sample syntax:

```
[form_param: "YourFieldName", count]
```

Specific values can be retrieved using a number parameter. For example, the number 2, formatted as "[form_param: "paramName", 2]," would return the second value associated with the field.

Additionally, the [loop: …] tag can be used to retrieve all of the form parameters associated with a field:

```
[loop: (form_param: "YourFieldName", count)]
   [form_param: "YourFieldName", loop_count]<br>
[/loop]
```

In the above example, "YourFieldName" is the name of the field that is the source of the count keyword.

Updating Multiple Records

The following is an example of the usefulness of the number parameter used with the [form_param: …] tag. In this example, multiple records appearing on a hitlist (or found set of records) can be updated by calling up a hitlist and submitting all records on one form to a page which then processes the individual updates. The response page that processes the actual updates, uses the [form_param: …] tag, [loop: …]…[/loop] tag, and [inline: …] operations to update each individual record.

To accomplish this, set up a hitlist as follows:

```
<form method="post" action="action.lasso?-nothing&
-response=process.txt">
[record]
   <input type="hidden" name="CurrentRecordNumber"
value="[recid_value]">
   <input type="text" name="YourFieldName1"
value="[field:"YourFieldName1"]" size=30>
   <input type="text" name="YourFieldName2"
value="[field:"YourFieldName2"]" size=30>
[/record]
</form>
```

The -nothing action sends the form submission to the page to be processed, in this case, it is titled "process.txt." The actual names of the fields to be updated in your database should be substituted for "YourFieldName1" and "YourFieldName2."

In order to direct the update to the correct record, the record ID must also be passed as a form parameter. In the above example, the value "CurrentRecordNumber" serves as a token, though it is not a Lasso tag and has no meaning. It simply allows the record ID value to be retrieved using a [form_param: …] tag. Note that the -token tag cannot be used in this case, since there are multiple record ID values and there can be only one token set with any one Lasso action.

The [loop: …] tag is used to individually process each instance of a form parameter value as it is substituted into an inline operation. The response page "process.txt" would process the series of record updates as follows:

```
[loop: (form_param:"CurrentRecordNumber",count)]
   [inline:
      database="YourDBName",
      layout="YourLayoutName",
```

```
        recid=(form_param:"CurrentRecordNumber",loop_count),
        "YourFieldName1"=(form_param:"YourFieldName1",loop_count),
        "YourFieldName2"=(form_param:"YourFieldName2",loop_count),
        update]
    [/inline]
[/loop]
```

The loop tag loops for the number of form parameters submitted (the number of records on the first page of the form). Each inline then submits the data to the database. There can be as many fields used as needed, with each appearing in the following format:

```
"YourFieldName2"=(form_param:"YourFieldName2",loop_count)
```

Values Paired with a Lasso Tag

As Lasso processes a response format file, the values associated with the current action can be substituted into the format file. These are generally values that were paired to the various Lasso command tags. Lasso will simply display the appropriate values, making them available to other fields, or allowing the values to be used with other Lasso tags.

For example, the substitution value can be placed as a value in a form:

```
<input type="hidden" name="field" value="[database_name]">
```

Or substituted into an embedded URL:

```
<a href="action.lasso?-search&-database=[database_name,url]&
-layout=[layout_name,url]&-response=detail.html&-recid=recid_value">
Link to Accomplish the Same Search </a>
```

Or within a conditional statement. For example, the [lasso_action] tag can be compared to the names of various actions, as follows:

```
[if: lasso_action=="search"] display one thing
   [else] ...display another
[/if]
```

In this manner, a returned HTML file can be dynamically altered depending on the previous search action.

Retrieving FileMaker Pro Database Info

The following tags gather information about the structure of FileMaker Pro databases. These tags are part of the Lasso

"Database_Info_Tags.mod" module. In order to use these tags, "Database_Info_Tags.mod" must reside in the "Lasso Modules" folder when Lasso is launched. Note: As there is a potential security risk using these tags in certain server configurations, the "Database_Info_Tags.mod" module is not installed by default.

Display Database Name

The [db_name: …]…[/db_names] container and its corresponding [db_names] sub-container retrieve the names of all open databases.

Display All Layout Names

The [layout_name: …]…[/layout_names] container and its corresponding [layout_names] sub-container retrieve the names of all layouts.

Display Field Name

The [field_name: …] tag is used to display the number of fields in a specified database, a database field's name, type, or accessibility (security privileges). The first parameter to this tag must be either a number or the keyword "count."

If "count" is specified, the total number of fields in the current layout will be returned. The syntax is as follows:

```
[field_name: count]
```

If there are ten fields in the specified layout, then the result is the number 10. Note that no actual field name is indicated.

A number is used to indicate the specific field in a layout in the "top to bottom" order it appears. For example, to retrieve the name of the first field in a layout, use the following:

```
[field_name: 1]
```

If used, a second parameter must be either the keyword "type" or "protection." The keyword "type" outputs either "text," "number," "image," "date/time," "boolean," or "unknown," depending on the type of data stored in the field. For example:

```
[field_name: 1, type]
```

The keyword "protection" will output "none" if the field is modifiable, or "read only" if the data cannot be modified (via an add or update).

```
[field_name: 1, protection]
```

The [loop: …] tag can be used with the [field_name: …] tag to return a list of every field in the database. For example, to display all fields in a layout and have them available in a search or add form, use the following:

```
[loop: field_name:count]<p><b>[field_name: loop_count]</b><br>
   <input type="text" name="[field_name: loop_count]" value="" size="60">
[/loop]
```

Hiding Non-Modifiable Fields

One needs to be aware that certain FileMaker Pro fields cannot be updated. This includes calculation, summary, auto-entered, validated, or fields that are not checked as "allow entry into field" in the "Field Format" dialog in FileMaker Pro. To accommodate this, one can either always reference layouts that contain modifiable fields, or use conditional statements to hide certain fields. For example:

```
[if: (field_name: 1, type)=="text"]
```

Or:

```
[if: (field_name: 1, protection)=="none"]
```

Loop

The [loop: …]…[/loop] container instructs Lasso to display the text between the opening and closing tags a specified number of times. This number can be a literal value or the result of another Lasso tag. The [loop: …] tag has limited functionality when used on its own; however, it can add looping functionality to other Lasso tags.

Here is an example of how the [loop: …] tag could be used to list all the field names in a specified layout:

```
[loop: field_name:count]<p><b>[field_name: loop_count]</b><br>
   <input type="text" name="[field_name: loop_count]" value="" size="60">
[/loop]
```

A loop can also be used to return all values for a certain field (if the field is repeating or has multiple values entered by a check box selection list). The following code:

```
[form_param: "group", count]<p>
[loop: (form_param: "group", count)]
      [form_param: "group", loop_count]<br>
   [/loop]
```

Returns the following result:

```
sales
printing
engineering
```

Loop Count

The [loop_count] tag outputs a number which represents the current repetition of items within containers such as [record], [value_list: …], [loop: …], [repeating: …], [search_args], [sort_args], or [portal: …]. For example, the first record in a portal would have a loop count of 1, the second 2, the third 3, and so on. Examples of how the [loop_count] tag is used follow.

```
[loop: "10"]
   This is loop # [loop_count].
[/loop]
```

Show all repeating values in a numbered list as follows:

```
[repeating:"YourFieldName"]
   [loop_count]) [repeat_value]<br clear=all>
[/repeating]
```

Number the rows in a portal as follows:

```
[portal:"jobs"]
   ROW #: [loop_count][field:"Jobs::Contacts"]<br>
[/portal]|
```

The [loop_count] can be used in conditional statements. For example:

```
[loop: "10"]
   [if: loop_count=="5"] This is the fifth time in the loop out of 10
[/loop]
```

The following will display the first five repetitions of a repeating field:

```
[repeating: "test"]
   [if: loop_count <= "5"]
      <input type="text" size=30 name="test" value="[repeat_value]"><br>
   [/if]
[/repeating]
```

The [loop_count] tag can also be used to place a comma after all [list_value] items, except the first, as follows:

```
[value_list: "Jobs"]
   [if: checked !=""][if: loop_count != "1"], [/if][list_value][/if]
[/value_list]
```

Not all instances of a field in a search form are displayed by the [search_args] and [sort_args] containers. Only coupled search arguments are returned. For example, using the database "Employees," a hitlist reply format file may contain the following syntax (outside of the [record] container):

```
no result:[database_name], [search_args] [loop_count]) [search_field]
[search_op] [search_value], [/search_args]
```

If the search form has 12 possible field entries and only the first, fourth and eighth were used in a search, the result would be as follows:

```
no result: Employees, 1) First Name begins with joanna, 4) Email begins with
feelin, 8) Comments contains groovy
```

Lasso Process

The [lasso_process: …] tag can be used to instruct Lasso to process any enclosed tags, and display the result. The [lasso_process: …] tag is especially useful when Lasso tags contained in a field are returned to a format file. For example, if you had a field named "result" containing tags, the following syntax could be used to process all tags contained in the field:

```
[lasso_process: field: result]
```

The result is the same as if the contents of the field were part of the original format file when it was processed.

HTML Comment

The [html_comment] container inserts the HTML comment tags.

```
[html_comment] Date is [server_date] [/html_comment]
```

Returns the following result:

```
<!-- Date is 11/21/97 -->
```

The [html_comment] container is particularly useful when working with JavaScript as it relies heavily on commented code.

Tags with Multiple Values

These container and substitution tags may contain multiple values:

[sort_args]…[/sort_args] — Substitutes the sort options used in the previous search. The [sort_field] and [sort_order] tags display values for each instance of the sort parameters used in the previous search and are only valid within the [sort_args]…[/sort_args] container. All HTML between [sort_args] and [/sort_args] is repeated for every field included in the search.

Chapter 18: Variables, Tokens, and Cookies

At times it is necessary to store and retrieve specific values that are not saved in a FileMaker Pro database. These values may be needed for tracking visitors to a site or providing customized responses based on current selections. Lasso provides three pairs of tags for this purpose: [set_var: …] and [var: …]; [token] and [token_value]; and [set_cookie: …] and [cookie: …]. All set a value and then allow the current value of the item to be retrieved via a substitution tag. Each approach handles information differently.

- Variables are used to set and retrieve values within the same processed format file.
- Tokens are used to set and retrieve a single value passed from format file to format file.
- Cookies are used to maintain a persistent value in the client's browser.

Unlike variables, token and cookie values are not available on the same processed format file from which they are set. Tokens can only have one value at a time, whereas multiple variable or cookie values can exist concurrently. In addition, variables and tokens are not stored in memory or a file, while a cookie is the only type of value that Lasso can set and retrieve from a location that is not a FileMaker Pro field.

Setting and retrieving values is mandatory for maintaining "state" or for tracking the identity of users to your Web site. Tracking users is required for e-commerce solutions and all situations where security is a concern. Note: As the concept of setting and retrieving a value pertains to security, it is important to note how Lasso Security compares with other methods for tracking users.

- Lasso Security is used to maintain a permanent value.

Declaring Variables

The value of a variable can only be used within the same processed file on which it is set. The [set_var: …] tag sets the value of the named variable. Any number of variables can be created with one [set_var: …] tag by pairing the name of the variable with

the value it is being set to. Each pair of variables is separated by a comma, with the name of the variable appearing to the left. The basic format of the tag is as follows:

```
[set_var: "var1"="SomeValue", "var2"="SomeOtherValue"]
```

The [var: …] tag retrieves the indicated variable for display or for use within another Lasso tag. If the specified variable has not been set, nothing will output.

Variables are quite useful for extracting the values of sub-containers or other values within container tags, in order to use these values outside of the container. Here are several examples where variables are of great benefit:

1. To retrieve a value that is a result of an inline action. For example:

   ```
   [inline: search, database="Primary", layout="main", operator="eq",
   "Name"=field:"userid"]
      [set_var: field:"Name"]
   [/inline]
   ```

   The value that results from the inline action can thus be used outside of the [inline: …]…[/inline] container.

2. Evaluate a conditional statement once and then use the result repeatedly throughout the processed file. For example:

   ```
   [if: var: "Open" >> field: "Category"]
      [set_var: "Open"="True"]
   [else]
      [set_var: "Open"="False"]
   [/if]
   ```

3. Set a variable to the value of a sub-container tag to allow this value to be used outside of the container tag. For example:

   ```
   [repeating: "CD"]
   [if: loop_count == "1"]
      [set_var: "1stRepeat"=repeat_value]
   [if: loop_count == "2"]
      [set_var: "2ndRepeat"=repeat_value]
   [/if]
   [/repeating]
   ```

   The first and second repeating values can then be used throughout the same format file, using [var: "1stRepeat"] and [var: "2ndRepeat"].

Using Variables With Conditional Statements

A conditional statement can be used to determine the value used for a certain variable, and then, allow that variable to be used elsewhere without having to check the condition again. The following example illustrates how variables can ease the processing of multiple actions on a reply page. Variables come in handy since two conditions are being checked: the action that called the page; and the name of the field to be used in an inline. This method reduces the number of files generated and simplifies creation and administration of a Web site.

First, a variable is set according to the type of action returned on the page. Nothing is displayed at this point. The syntax is as follows:

```
[If: lasso_action=="add"]
   [set_var: "textaction="Added"]
[else: If: lasso_action"=="update"]
   [set_var: "textaction"="Updated"]
[else: If: lasso_action"=="delete"]
   [set_var: "textaction"="Deleted"]
[else]
   [set_var: "textaction"="search"]
[/if]
```

Second, a conditional statement is used to determine the name of the database used in the previous action:

```
[if: lasso_action=="search"]
   [include: "Detail.txt"]
[else]
   [If: database_name=="Jobs"]
      <font size="+2" color="#ED181E">
      <h2 align=center>Job [var: "textaction"]</h2>
      </font><hr>
      [set_var:"fieldval"=field:"Jobname"]
   [else: if: database_name=="Tasks"]
      <font size="+2" color="#ED181E">
      <h2 align=center>Task [var: "textaction"]</h2>
      </font><hr>
      [set_var:"fieldval"=field:"Task Name"]
   [else]
   ERROR
   [/if]
```

```
[inline: search, database="Primary", layout="main", operator="eq",
"Name"=var:"fieldval"]
   [include: "Detail.txt"]
[/inline]
[/if]
```

In the above example, two different fields were used in two related databases ("Jobs" and "Tasks"). If an action is completed in one of the databases, the file will determine how to show the detail for the entire main record (from the "Primary" database). The inline cannot contain a conditional statement to determine which field to specify. Thus, another variable is used to set the name of the field used in the inline. The first condition uses the included file as is, since the file can be correctly called by a search action.

In the next example, a variable is used to resort a found set of records according to new sort options. This is illustrated in the provided Employees example in the "Hitlist2.html" format file. A variable is used here to capture the first field used in a sort, as follows:

```
[sort_args][if: loop_count == "1"]
[set_var: "sortedfield"=sort_field]
[set_var: "sortedby"=sort_order]
[/if][/sort_args]
```

This variable is later used to pre-select the last selection in the sort field as well as sort by pop-up list selections. This allows one to place the value of the first sort field as the selected item, so that it appears as selected. The variables can be displayed as follows:

```
<p>The variable "Sorted Field" contains [var: "sortedfield"]
<p>The variable "Sorted By" contains [var: "sortedby"]
```

The variables can also appear within a pop-up list to show the selections made the last time the displayed list was sorted. For example:

```
<b>Sort by :</b>
      <select name="-sortfield">
         [if: var:"sortedfield"==""]
         <option value="" selected>- Select -
         <option>unsorted
         [else: if: var:"sortedfield"=="unsorted"]
         <option selected>unsorted
         [else]
         <option value="[var: sortedfield]" selected>[var: "sortedfield"]
         <option>unsorted
```

```
        [/if]
        <option>First Name
        <option>Last Name
        <option>Employee Number
        <option>Hire Date
        <option>Group
        <option>Shift
      </selected>

<select name="-sortorder">
   [if: var:"sortedby"=="descending"]
   <option>ascending
   <option selected>descending
   <option>custom
   [else: if: var:"sortedby"=="custom"]
   <option>ascending
   <option>descending
   <option selected>custom
   [else]
   <option selected>ascending
   <option>descending
   <option>custom
   [/if]
</select>
```

Tokens

A token is used to pass a value from one format file to the next without being saved in a database. This is accomplished by pairing a -token tag with a value and then allowing this to be included in the "name=value" pairs that make up a post or search argument. A new token is set using the -token command tag. The token is retrieved on a reply page using the [token_value] tag. If the token was set on the first page, the value of that token will be substituted wherever [token_value] appears. To pass the token value to subsequent format files, the token needs to be reset.

What are tokens used for? A token (like a cookie) can facilitate the creation of shopping cart applications, as for example, when it is necessary to utilize a customer ID or order ID on every form throughout a shopping session in order to keep track of what a user orders. A token can also carry a value that is used in an [if: …] tag in the reply page to direct a conditional response.

Set Token

A token is typically set to either the contents of a FileMaker Pro field, an entered value, or to the record ID (to keep track of a particular record in the FileMaker Pro database). It could also be a literal value, though a token is limited to 255 characters. The syntax for setting a token to be included with a form submission is:

```
<input type=hidden name="-token" value="some value">
```

For example:

```
<input type=hidden name="-token" value="[recid_value]">
```

When using a field value it would take the following form:

```
<input type=hidden name="-token" value="[field:"YourFieldName"]">
```

When the token is to be entered on a form using a standard input field, the following is used:

```
<input type="text" name="-token" size=30>
```

It can also be selected from a pop-up as follows:

```
<select name="-token" size=1>
    <option>token value1
    <option>token value2
    <option>token value3
</select>
```

Tokens can also be set with a Lasso call in a link. For example:

```
<a href="action.lasso?-database=Products.fp3&-layout=Layout1&
-token=[recid_value]&[response]=productlist.html&-show">Browse our
products</a>
```

The token must be set before the [token_value] can be displayed on the response file.

Display Token with [token_value]

Lasso will substitute the value for the token into a format file whenever the [token_value] tag appears. The token value is displayed on the response file by simply placing the tag as you would any field returned by Lasso:

```
Display Token: [token_value]
```

The token can be inserted into a FileMaker Pro field as follows:

```
<input type=hidden name="YourFieldName" value="[token_value]">
```

The [token_value] tag takes the optional parameters "raw" and "url" which perform the same functions as with the [field: …] tag. This is demonstrated as follows:

```
<input type=hidden name="-token" value="[token_value,raw]">
```

To pass the token, it must be set on each subsequent format file which calls a Lasso action, except that Lasso will automatically copy the value into URLs automatically created with the [detail_link: …], [next], and [prev] tags.

Using Tokens

The following represents how to pass record IDs via a token. In this example, the token is set to the current record ID when any action is submitted:

```
<input type=hidden name="-token" value="[recid_value]">
```

To continue to pass this value to another file, the token needs to be set again using:

```
<input type="hidden" name="-token" value="[token_value]">
```

Or within an embedded URL as:

```
<a href="action.lasso?-database=Products.fp3&-layout=Layout1&
-token=[token_value]&[response]=productlist.html&-show">
Browse our products</a>
```

If the value of the token is needed within an inline statement on the directly returned response format file, it needs to be set again. The following syntax is used to set a token within an inline:

```
[inline: database="YourDBName", layout="YourLayoutName",
token=token_value, search]
   Here is the token value: [token_value]
[/inline]
```

If a token should continue to be passed from file to file, then it must be set on each file or retrieved. The token can be changed on new files by setting it to a new value.

Multiple Tokens

Lasso can only set one token with any Lasso action. However, it is possible to pass multiple tokens in an indirect way by concatenating several values and setting the token to this value. The concatenated value should include a delimiter so the tokens can be extracted on the reply page using the [string_getfield: …] tag.

For example, on a form you can take some values (literal, field, calculated, and so on) and set it to a token as follows:

```
<input type="hidden" name="[token]" value="[string_concatenate: "value1","#",
" value2", "#", token_value]"
```

Note that a pound sign is used as a delimiter.

On the reply page, the first value ("value1") can be retrieved using:

```
[string_getfield: token_value, field_number=1, delimiter="#"]
```

The second value can be retrieved using:

```
[string_getfield: token_value, field_number=2, delimiter="#"]
```

The third value can be retrieved using:

```
[string_getfield: token_value, field_number=3, delimiter="#"]
```

Note: The [string_getfield: …] tags require the "String_Tags.mod" module to be installed in the "Lasso Modules" folder.

Cookies

Cookies allow data to be saved in the browser settings of the computer accessing your Web page, and for that data to be retrieved within the same session or at a later time. That is, if the cookie hasn't been purged from the user's computer. Lasso has the ability to set HTTP cookie values using the [set_cookie: …] tag in any Lasso format file. The cookie value is retrieved using the [cookie: …] tag.

Netscape defines cookies as follows:

> "Cookies are a general mechanism which server side connections (such as CGI scripts) can use to both store and retrieve information on the client side of the connection. The addition of a simple, persistent, client-side state significantly extends the capabilities of Web-based client/server applications."

For more details about cookies, refer to Netscape's description at:

```
http://home.netscape.com/newsref/std/cookie_spec.html
```

Note: Cookies cannot always be relied upon as some browsers provide an option to not save cookies.

Setting an HTTP Cookie

Setting a cookie saves a piece of data relevant to the visitor to the Web site which can be recalled for subsequent visits. The components for setting a cookie are as follows:

```
[set_cookie: "cookie_name"="cookie_value", expires="minutes_from_now",
path="path_name", domain="domain_name",secure]
```

For example:

```
[set_cookie: "Login"=field:"userlog",expires=20000]
```

The "cookie_name" and "cookie_value" parameters are required and must appear first in the parameter list immediately following the [set_cookie: …] tag:

- **cookie_name —** The cookie name must be less than 1024 characters. It can be either a literal or a field value. Specify a field value as follows:

  ```
  field: "YourFieldName"
  ```

  Netscape prohibits the use of the semi-colon, comma, or space characters in cookie names. However, Lasso performs URL-style encoding and decoding of cookie names, so these characters may be used.

- **cookie_value —** This is the data value for the cookie, and must be less than 1024 characters. It can be either a literal or a field value. Record IDs can be used by specifying [recid_value] for the cookie value.

The following parameters are optional and may appear in any order. Here's a description of each parameter for the [set_cookie: …] tag:

- **expires** — Determines the number of minutes in the future until the cookie will expire. If not set, the cookie will expire at the end of the user's browser session (when they quit their browser). "minutes" must be less than 10 digits (less than 1 billion). A value of 0 (zero) or a negative number may be used to have cookies expire immediately.

- **path** — This sets the path attribute for the cookie. "path_name" must be less than 256 characters. If not specified, the path defaults to the path of the active URL (at the time when the format file with the [set_cookie: …] tag is processed). The cookie can only be returned to a Lasso response page when the domain name and path match that stored when the cookie was originally set. To set the path to the root level of the Web server indicate a slash as the path to the cookie:

  ```
  [set_cookie: UserName=field:name,path="/", expires=10000]
  ```

- **domain** — This sets the domain name attribute for the cookie. "domain_name" must be less than 256 characters. If not set, the domain defaults to the domain of the server.
- **secure** — With "secure" specified, the cookie is transmitted only over secure communications channels using the HTTPS (HTTP over SSL) protocol.

The total number of characters between the opening and closing square brackets of the [set_cookie: …] tag must be less than 2048.

Displaying a Cookie Value

After a cookie is set it is returned in every subsequent request in which the "path_name" and "domain_name" match those specified in the [set_cookie: …] tag. The cookie data value may be displayed in a Lasso format file via the [cookie: …] command:

```
[cookie: "YourCookieName"]
```

The "cookie_name" is the same name specified in the [set_cookie: …] tag.

The [cookie: …] tag accepts the optional "url" parameter to perform URL-encoding:

```
[cookie: "YourCookieName",url]
```

Cookies may be used in conditional [if: …] tags. For example:

```
[if: cookie: cookie_name operator comparison_value]
    …HTML if condition is true…
[else]
    …HTML if condition is false…
[/if]
```

The "cookie_name" is the same name specified in the [set_cookie: …] tag. The "operator" may be any of the standard operators supported by the [if: …] tag. The "comparison_value" can be either a hard-coded literal value or a field name in the form "field: field name" (without the quotes and no square brackets). The "operator" determines how the comparison is made and is used in the short form.

Tip: When testing cookies in a format file, it is helpful to turn on the cookie alert in the browser. You can then easily see when they are added.

Chapter 19: Math and String Tags

Math Tags

The math tags perform calculations on numeric values returned to a format file. The following guidelines apply:

- Each value must be a number, or any Lasso tag that is substituted as a number.
- Each value is separated by a comma.
- Quotes can be placed around each literal value, but are not necessary as only numeric values apply.
- In all math tags, except the [math-round: …] tag, parameters specified as whole numbers always output a whole number. For example, when using [math-div: …] to divide 5 by 2, the output is 3 (2.5 rounded up), but when dividing 5 by 2.0 the output is 2.5. Additionally, the precision of the output (the number of digits after the decimal point) is inherited from the parameters themselves. For example, when adding 1, 1.0 and 1, the output will be 3.0, but when adding 1, 1.00 and 1.000, the output will be 3.000.

In order to use these tags, the "Math_Tags.mod" file must reside in the "Lasso Modules" folder when Lasso is launched.

The math tags include the following:

[math-add: …] — Adds a list of values. For example:

> [math-add: 2, 2, 14, 3] outputs to 21 (it is equivalent to: 2 + 2 + 14 + 3).
>
> [math-add: field:"price", 6.3] outputs to 26.3, if the current value of the field "price" is 20.

[math-sub: …] — Subtracts a list of values. The specified values are subtracted in the order in which they appear. For example:

> [math-sub: 20, 1] outputs to 19, and [math-sub: 5, 3, 20, 1] outputs to -19.

[math-mult: …] — Multiplies a list of values. For example:

> [math-mult: 5, 5, 14, 2] outputs to 700 (it is equivalent to: 5 * 5 * 14 * 2).

[math-div: …] — Divides a list of values. The specified values are divided in the order in which they appear in the string. For example:

```
[math-div: 25, 5, 2] outputs to 3
[math-div: 25, 5, 2.0] outputs to 2.5
```

[math-mod: …] — Divides two values and outputs the division remainder. The remainder is also referred to as the modulo. For example:

```
[math-mod: 29.50, 5] outputs to 4.5 (equivalent to: 29.50/ 5 = with a remainder of 4.5)
```

[math-round: …] — Rounds a value to the nearest value. Only one value can be evaluated with one [math-round] tag. A number parameter can be used to specify the precision of the rounded result. The number of decimal places to be used is indicated by the tenth power, i.e., 100, 10, 1, .01, .001, and so on. For example:

```
[math-round: 1345.75, 1] outputs to 1346 (equivalent to: 1345.75 rounded to the nearest 1)
```

Nesting Math Tags

Complicated expressions can be created by nesting math tags within parentheses. For example:

```
[math-sub: (math-add: 2, (math-div: (math-mult: 2, 6), 4)), 2]
```

Lasso will output the number 3 (this is equivalent to: 2 + 2 * 6 / 4 - 2).

Creating a Page Counter

In this example the [math-add: …] tag is used to increment a field by one, every time the record is displayed on a detail page. This example is demonstrated in the provided "Employees" database example. The value for the number field "visitcount" is returned to the [math-add: …] tag, incremented by 1. An inline then adds the value back to the database as follows:

```
[if: lasso_action=="search"]
[inline: update, database=database_name, layout=layout_name,
   visitcount=(math-add:(field:"visitcount"),1), recid=recid_value]
   This record has been visited <b>[math-add: (field:"visitcount"), 1]</b> times.
[/inline][/if]
```

Note that a conditional statement is used to ensure that the inline operation occurs only when the last action is search.

String Tags

The Lasso "String_Tags.mod" module provides many text parsing and manipulation capabilities. In order to use these tags, the "String_Tags.mod" file must be reside in the "Lasso Modules" folder when Lasso is launched.

The string tags include the following:

[string_concatenate: …] — Concatenates any number of unnamed string parameters.

```
[string_concatenate: "StringText", "MoreStringText"]
```

[string_countfields: …] — Returns the total number of fields in all of the specified text strings together.

```
[string_countfields: delimiter="DelimiterText", "StringText", ...]
```

[string_extract: …] — Extracts a specified range of characters from the source string.

```
[string_extract: start_position=#, end_position=#, "StringText"]
```

[string_findposition: …] — Returns the numeric position of the beginning of specified text.

```
[string_findposition: find="FindText", "StringText"]
```

[string_getfield: …] — Returns the text value of a field.

```
[string_getfield: field_number=#, delimiter="DelimiterText", "StringText" ]
```

[string_insert: …] — Inserts the specified string into the source string at a specified position.

```
[string_insert: position=#, text="InsertText", "StringText"]
```

[string_length: …] — Totals the number of characters of a specified string.

```
[string_length: "StringText"]
```

[string_lowercase: …] — Concatenates all specified text and returns it in lowercase.

```
[string_lowercase: "StringText"]
```

[string_remove: …] — Removes a specified range of characters from the source string.

```
[string_remove: start_position=# end_position=#, "StringText"]
```

[string_removeleading: …] — Removes all occurrences of a specified pattern which are found at the beginning of specified parameter strings.

```
[string_removeleading: pattern="PatternText", "StringText", ...]
```

[string_removetrailing: …] — Removes all occurrences of a specified pattern which is found at the end of the parameter strings.

```
[string_removetrailing: pattern="PatternText", "StringText", ...]
```

[string_replace: …] — Replaces specified text with specified replacement text.

```
[string_replace: find="FindText", replace="ReplaceText", "StringText"]
```

[string_uppercase: …] — Concatenates all specified text and returns it in uppercase.

```
[string_uppercase: "StringText"]
```

Chapter 20: Sending Apple Events

Lasso offers the ability to send Apple Events to other applications (including the Finder) from a processed format file. This enables one to control/automate processes independent of FileMaker Pro. For example, one may send Apple Events to MacAuthorize for credit card authentication.

Apple Events are configured and sent using the [event: …] tag. See CHAPTER 3: INSTALLATION AND SETUP for information on how to enable and disable [event: …] tag.

Note: Due to potential security concerns with certain configurations, the [event: …] tag is not installed by default.

There are several parameters needed to send an Apple Event. The [event: …] tag is constructed as follows:

```
[event: class="CLas", id="ThID", target="TargetApp", wait_reply=true]
   event_string
[/event]
```

Here is one example that instructs the application "UserLand Frontier" to beep once, then display the message "Hello, World!":

```
[event: class="misc", id="dosc", target="LAND", wait_reply=false]
"----":“speaker.beep(); msg("Hello, World!")”
[/event]
```

Note that some of the quotes shown within the event string are curly quotes (option-[and shift-option-[ ), as required by Apple Events:

```
[event: class="misc", id="dosc", target="LAND", wait_reply=false]
   "----":“speaker.beep(); msg("Hello, World!")”
[/event]
```

Event Tag Parameters

The [event: …] tag takes the following parameters:

- **class —** Every Apple Event has a class. Apple Events are grouped by their class. Several different Apple Events may have the same class.
- **id —** Each type of Apple Event will have a unique ID.

- **target —** This parameter specifies the entity to which the event will be sent. The target application is specified with either the creator code of the application or the exact name as it appears under the application menu.
- **wait_reply —** Instructs Lasso to either wait for the reply from the target application (=true), or proceed processing immediately after sending the event (=false). If Lasso does not wait for the reply, whatever data the target application may have returned will not be available for use later.
- **event_string** — Is the text of the event. The text between the [event: …]…[/event] tags specifies exactly which parameters should be included in the event. "----" signifies the direct object of the event. Any Lasso tags can be used with the "event_string."

  The "event_string" lets Lasso send events of great complexity, including complex object descriptors, using only text. This text should follow the format as specified by the AEGizmos Apple Event utility library. This format was chosen as it clearly specifies a manner in which the full power of Apple Events, including complex object descriptors, may be utilized within the confines of text.

  Here is an example of an event used to ask FileMaker Pro for the name of the first database:

```
[event: class="core", id="getd", target="FileMaker Pro", wait_reply=true]
   "----":obj {form:prop, want:type(prop), seld:type(pnam),
   from:obj {form:indx, want:type(cDB ), seld:long(1),
   from:"null"()} }
[/event]
```

  If this example appears difficult, that's because it is! This complexity is unavoidable when dealing with data structures like Apple Events. All the standard Lasso tags can be used within the "event_string." Here is an example of how FileMaker Pro is queried for the number of records in the database that was just searched:

```
[event: class="core", id="getd", target="FileMaker Pro", wait_reply=true]
   "----":obj {form:name, want:type(cDB ), seld:"[database_name]",
   from:"null"()} , kocl:type(crow)
[/event]
```

Post–Apple Event Tags

After the event is sent, there are three tags that can be inserted in the page, or used within [if: …] conditional statements:

- **[event_resultcode] —** This determines the result of the event. 0 (zero) indicates the event was sent successfully. A value other than 0 is the actual code returned from the Apple Event manager function which generated the error. This tag is available even if "wait_reply=false." This will indicate that the event was sent successfully.
- **[event_errorstring]** — If an error occurs, most applications will provide a description of what went wrong. These errors are reported using the [event_errorstring] tag. This tag is available if "wait_reply=true" and [event_resultcode] is not 0 (zero). The description of the error is provided by the target application. Some target applications will not supply this variable, in which case it will evaluate to a blank string.
- **[event_result]** — This tag inserts the result of the event into the page. This is available only if "wait_reply=true."

  If another event is sent, these tag values are overwritten with the new values for the new event. If these tags are used before an event is sent, they will evaluate to blank values.

  [event_result] can be coerced into a string with a variable number of parameters. Parameters are available only if "wait_reply=true." The parameters should be either a number or an Apple Event keyword. If the Apple Event result is a list or a record, these parameters are used to extract a specified list item by name, or record item by key. For example:

  ```
  [event_result, 2, 5]
  ```

  …will extract the fifth item of the second list.

  If the result was an AERecord, then the item with the "form" key should be extracted:

  ```
  [event_result, "form"]
  ```

  And…

  ```
  [event_result, 1, 3, 7, 2]
  ```

  …would extract item two from the seventh item of the third list of the first item of the reply list.

Chapter 21: Java-Enabling FileMaker Pro

Java Overview

Java is a fairly new programming language which has received a lot of interest recently. It was designed to be fully object oriented, i.e., modern. It is also totally cross-platform, which means one can compile a Java program and be able to run it unmodified on any computer platform for which there is a Java run-time or "Virtual Machine." It is mainly for this reason that Java has become the method of choice for creating and distributing online applications through the World Wide Web. All that is required to view standard Java applets, or mini-applications, over the Web is a Java-enabled Web browser. Many Web sites today are combining common HTML with Java applets to create richer and more compelling experiences for visitors. It is not uncommon to visit sites with scrolling Java banners, interactive Java image maps or chat rooms created completely in Java.

Java offers many advantages to the site designer over straight HTML. It allows one to create much more sophisticated interfaces than possible with static HTML. With Java, one does not have to be limited to the standard buttons and text fields provided by HTML. One can create sliders, active image buttons or other controls which can fit the intended purpose better than the common radio, check box or push buttons. Interfaces can be created which can quickly change from displaying the information in a tabular format to displaying it as a bar chart with the click of the button. The user would not have to wait for the page with the new view to reload or for images to be created by CGIs on the server side. Interfaces such as this would be extremely difficult or impossible to develop using static HTML.

The site designer may experience some problems as they Java-enable their site. Sites which utilize many Java applets may force their users to sit through long wait times as the applets are downloaded. Java also may require the Web browser to consume more memory and may also require faster, higher-end computers to run certain applets than static pages using common HTML. This may limit the number of potential viewers of the site to those with more powerful computers (or more patience). Also, not every

browser supports Java. This prevents those not using the more popular Java-enabled browsers from fully experiencing a site which is Java enhanced.

Java applets are generally much more complicated to create than HTML files. A site designer can quickly create an HTML-based site using nothing more than a text editor. To create Java applets, however, requires that the code be compiled into class files using either a commercial Java development environment or the standard Java Development Kit distributed by Sun Microsystems. Either way, this process can be more time consuming than creating common HTML pages and requires that the applet creator learn the Java language and the standard Java class library before s/he can truly begin creating Java-based sites. Because of the steeper learning curve involved with Java, many site designers will be forced to outsource their applet creation, which can lead to greater expense.

There are many compelling advantages for the programmer to use Java over languages such as C or C++. If C++ programmers want to create cross-platform applications, they are forced to use a cross-platform framework or to completely recreate their work for any additional platforms they wish to support. This process can be very time consuming and often the platform specific versions contain inconsistencies or code differences which can lead to support costs and lost stability. Java does not have these problems because it has a "compile once, run everywhere" philosophy. The programmer can write the Java application once, compile it and expect it to run on any platform which supports a Java Virtual Machine.

Java has many of the built-in features which are required for modern applications, yet are not included in the standard C or C++ language libraries. For instance, Java has built-in functionality to display common image formats such as GIF or JPEG. It also has integrated support for multi-threading and synchronizing critical sections of code. It also has TCP and UDP networking classes which make Java an Internet-savvy language. These classes make it trivial to retrieve a file from a Web site or to download a file from an FTP server. This functionality makes Java the language of choice for distributed network applications. Java also has a robust security model which prevents applets from reading files from the user's hard drive or sending data to servers other than the one the applet has come from. This can give users a sense of security when using applications which originate and communicate via the Internet.

Despite its advantages, Java is generally slower than an application compiled with C or C++. This means it is not suited for high performance applications which are graphically or computationally intensive. JIT or "Just In Time" Java compilers help somewhat by dynamically re-compiling the Java applications into native machine code as they run. This makes the Java applications much faster than the non-JIT versions. Java's cross-platform standard also comes with a dark side. Because Java code is the same for every platform, a Java application cannot take advantage of any special functionality supported by regular applications of that platform. For instance, Drag-and-Drop, supported by many Macintosh applications, is not available to Java applets or applications because it is not supported in the same way on other platforms.

Lasso and Java

Lasso supports serving information from FileMaker Pro databases to Java applets. This means the applet creator can utilize the power of Lasso and FileMaker Pro to create sophisticated solutions which are not confined by the limitations of HTML. Applets can utilize multiple databases simultaneously and can present the information in ways not possible with HTML. Applet programmers can take advantage of this new feature by utilizing the LassoProxy classes. The LassoProxy classes are a set of Java source files which can be compiled with Java applets. The classes know how to formulate Lasso requests, send them to the server where Lasso is running, and make the results available for the applet to use. By using this API, applets can perform any of the database activities possible with HTML forms. They can perform finds, updates deletes and can create new records in FileMaker Pro databases.

The LassoProxy classes communicate with Lasso using standard HTTP. They formulate a request string based on the parameters the applet programmer selects, and then the classes send the request to the server. Lasso then returns the response, but instead of returning it as HTML as it would normally, it sends only the data and lets the applet choose how to display it. Applets can elect to receive only certain fields from the database, making the response very compact and efficient.

There are three Java classes programmers can use to communicate with Lasso:

- LassoRequest
- LassoProxy
- LassoResponse

The first class, LassoRequest, is used to set up the action Lasso will take. Using just a few of the classes' methods, the programmer can specify the database, layout and fields used for the request.

The following Java snippets show how to set up a request to perform a search:

1. Create a new Lasso Request object.
    - **LassoRequest**
      request = new LassoRequest();
2. Assign the database name and layout name to be used in the search.
    - request.setDatabaseName("MyDatabase");
    - request.setLayoutName("MyLayout");
3. Specify the action to be taken.
    - request.setAction(LassoRequest.SEARCH);
4. Add the fields to be searched and the values to be searched for along with the search operators to be used for each field.
    - request.addField("Field_One", "searchvalue",
      LassoRequest.EQUALS);
    - request.addField("Field_Two", "anothervalue",
    - LassoRequest.BEGINS_WITH);

The search request is now completely set up and is ready to be processed. Additional options are available to the programmer such as setting the maximum number of records to be returned, setting the logical operator (AND or OR) on a global or field level, specifying a timeout or sort criteria. All of these operations can be performed with ease using only a few additional lines of code.

When a request is ready to be processed, the second class, LassoProxy, is required. The LassoProxy class takes a LassoRequest object, sends the request data to the server where Lasso is running, and receives the response from the server. The LassoProxy object then converts the response data into a LassoReponse object which the programmer can use to retrieve the search results. All of this processing happens behind the scenes. The programmer only needs to use one of the LassoProxy classes methods to process the entire request.

To process the request object created earlier, the following steps are required:

1. Create a LassoProxy object.

   The LassoProxy object requires a URL object to be created. The getDocumentBase() method returns a URL object representing the server the applet was downloaded and run from.

   - **LassoProxy**
     proxy = new LassoProxy(getDocumentbase());

2. Have the LassoProxy object process the request.

   This will return a LassoResponse object. Here, the LassoProxy object is passed the LassoRequest object (called "request"). Within the LassoRequest object is the search requested created earlier.

   - **LassoResponse**
     response = proxy.processRequest(request);

Now that the request has been sent to Lasso and the results have been returned, the programmer can use the LassoResponse class to retrieve this information. The information which the programmer has access to includes:

1. a result code indicating if the request was successfully processed
2. the number of records returned
3. the total number of records in the database
4. the names, types and value lists (if any) for each of the fields on the specified layout and the data
5. record IDs for the returned records

All of this information can be retrieved using the various methods belonging to the response object. Below are some examples of how the data can be retrieved.

All examples assume a LassoResponse object called "response" was created earlier.

1. Retrieve the names of all the fields in the previously specified database. The names are returned as an array of strings. Standard Java code can be used to further process this array as

required. This snippet will print the number of fields and their names to the standard output.

```
String names = response.fieldNames();
System.out.println("There are " + names.length +
            " fields in the database:");
for (int i = 0; i < names.length; ++i)
   System.out.println("\t" + names[i]);
```

2. Retrieve the types of data the fields can hold.

   Each field in the database is defined to hold a particular type of data.

   Some hold text, some hold dates or times and some can hold pictures. The following snippet prints the type of each field to the standard output.

```
for (int i = 0; i < response.numFields(); ++i)
{
   System.out.print("Field " + response.fieldNames()[i] +
               " is a ");
   switch(response.fieldType(i))
   {
      case LassoResponse.CHAR:
         System.out.println("CHAR");
         break;
      case LassoResponse.SHORT_INTEGER:
         System.out.println("SHORT_INTEGER");
         break;
      case LassoResponse.INTEGER:
         System.out.println("INTEGER");
         break;
      case LassoResponse.SHORT_FLOAT:
         System.out.println("SHORT_FLOAT");
         break;
      case LassoResponse.FLOAT:
         System.out.println("FLOAT");
         break;
      case LassoResponse.IMAGE:
         System.out.println("IMAGE");
         break;
      case LassoResponse.DATE_TIME:
         System.out.println("DATE_TIME");
         break;
```

```
        case LassoResponse.BOOLEAN:
            System.out.println("BOOLEAN");
            break;
        default:
            System.out.println("strange type");
            break;
    }
}
```

3. Retrieve the data from the returned records.

   The following snippet will print the data in each field for each of the returned records to the standard output.

```
System.out.println("Here is the returned data:");
// loop for each record
for (int recordNum = 0; recordNum < response.numRecords(); ++recordNum)
{
    System.out.println("\tRecord number " + recordNum + ":");
    // loop for each field
    for (int fieldNum = 0; fieldNum < response.numFields(); ++fieldNum)
    {
        System.out.println("\t\tField name: " +
                           response.fieldNames()[fieldNum]);
        // loop for each value the field may contain
        for (int valueNum = 0;
         valueNum < response.fieldData(recordNum, fieldNum).length;
         ++valueNum)
        {
            System.out.println("\t\t\t" +
                           response.fieldValue(recordNum,
                                               fieldNum,
                                               valueNum));
        }
    }
}
```

4. More examples.

   More examples can be found distributed with Lasso. These examples show how to retrieve information about databases and how to search, add, update and delete records using the LassoProxy API.

The LassoProxy API Reference

Here are the classes in the LassoProxy API, the public methods supported by each, and a short description of how each one functions:

```
class LassoRequest
    LassoRequest(String lassoName)
        // Construct a LassoRequest object and set the Lasso name.
        // This is the name used to reference the Lasso application running on
        the server.
        // For example: action.lasso or Lasso.acgi
    LassoRequest()
        // Construct a LassoRequest object using the default Lasso
name: action.lasso
    void resetAll()
        // Resets the LassoRequest object to its initial state
        // Removes all search fields, sort fields and sort orders,
        // the database and layout name and sets the action to SEARCH
    void resetSearch()
        // Same effect as resetAll(), but does not remove the database or layout
        name
    void resetSort()
        // Removes all the sort fields and sort orders
    int numSearchFields()
        // Returns the number of search fields that have been added to the request
    void setDatabaseName(String name)
        // Sets the name of the database the request will operate on
    void setLayoutName(String name)
        // Sets the name of the layout the request will operate on
    String databaseName()
        // Returns the name of the database the request will operate on String
        layoutName()
        // Returns the name of the layout the request will operate on void
        setAction(int value) throws IllegalArgumentException
        // Sets the action for this request
        // Possible values are:
        //
        // LassoRequest.SEARCH
        // LassoRequest.ADD
        // LassoRequest.UPDATE
        // LassoRequest.DELETE
        // LassoRequest.FIELD_INFO
        // LassoRequest.FIND_ALL
        // LassoRequest.RANDOM
```

```
    //
    // The default action is SEARCH
int action()
    // Returns the action for this request
void setMaxRecords(int value)
    // Set the maximum number of records which will be retrieved at once
    // The default value is 50
    // To retrieve all the records at a time, specify LassoRequest.ALL
int maxRecords()
    // Returns the maximum number of records which will be retrieved
    // at a time
void setRecordID(int value)
    // Set the id of the record this request will operate on
    // This is required when the action is set to UPDATE or DELETE
int recordID()
    // Returns the record ID for this request
void setSkipRecords(int value)
    // Set the number of records Lasso will skip when retrieving the
    // results. This is used to retrieve a certain number of records at a time.
int skipRecords()
    // Returns the skip records value for the request
void setLogicalOperator(int value)
    // Set the global operator for the request
    // This is only used when the action is set to SEARCH
    // Possible values are:
    //  LassoRequest.AND
    //  LassoRequest.OR
    //
    // The default value is LassoRequest.AND
int logicalOperator()
    // Returns the global logical operator for this request
void setTimeout(int value)
    // Set the maximum number of seconds Lasso will wait
    // for a response from the database
int timeout()
    // Returns the timeout value for the request
void addReturnField(String fieldName)
    // Specify a field which will be returned in the response.
    // By default, all fields will be returned.
    // Applets can use this method to limit the amount of data
    // returned to only that with which are concerned.
void addField(String fieldname, String value)
    // Add a field value by name
    // This is used when the action is SEARCH or UPDATE
    // When the action is SEARCH, this specifies a field to be searched
```

```
        // and the value to be searched for. The default field operator,
BEGINS_WITH is used.
        // When the action is UPDATE, this method specifies a fields new value.
        // To specify multiple valueS for a field (a field with a value list for
instance),
        // call this method once for each value to be added.
    void addField(String fieldName, String value, int fieldOperator)
        // Add a search field with a specific field operator
        // Possible value for the field operator are:
        // LassoRequest.EQUALS
        // LassoRequest.NOT_EQUALS
        // LassoRequest.CONTAINS
        // LassoRequest.BEGINS_WITH
        // LassoRequest.ENDS_WITH
        // LassoRequest.GREATER_THAN
        // LassoRequest.GREATER_THAN_EQUALS
        // LassoRequest.LESS_THAN
        // LassoRequest.LESS_THAN_EQUALS
    void startOperator(int op)
        // Specify the beginning of a logical grouping of search fields.
        // Fields added after calling this will be searched using the specified
        operator.
        // For instance, one can specify several fields to be searched as OR
        // and several other fields to be searched as AND
        // These two groups can in turn be combined together as OR
    void endOperator()
        // Specify the end of a logical grouping of fields.
        // Search fields added after calling this will resume being grouped under the
        // previous logical operator (or the global logical operator, which defaults
        to AND)
    boolean validFieldOperator(int fieldOperator)
        // Returns that the operator specified is valid
    boolean validSortOrder(int order)
        // Returns that the specified sort order is valid
    String toString()
        // Returns the parameter string which can be submitted to Lasso
        // This method is used by the LassoProxy class and programmers
        // will probably never need to access this. However, this method is
        // made public for those with special needs.
class LassoProxy
    LassoProxy(URL baseURL, String lassoName)
        // Constructs a LassoProxy object and specifies the URL to which
requests will be submitted
        // Also specifies the Lasso name, e.g., "action.lasso" or "Lasso.acgi"
    LassoProxy(URL baseURL)
```

```
        // Constructs a LassoProxy object which used the default
Lasso name: "action.lasso"
    String[] databaseNames()
        // Returns an array of all the databases open of the server
    String[] layoutNames(String databaseName)
        // Returns an array of all the layouts in the specified database
    Image getJPEG(LassoRequest request, String fieldName,
                int bitsPerPixel, int imageQuality)
        // Retrieves an image from the database in JPEG format.
        // The LassoRequest object must have the database name, layout name
        // and the record ID set accordingly. fieldName specifies the field in the
        // database where the image resides. bitsPerPixel is either 16 or 32 and
        // imageQuality is a value from 0 through 4 and signifies the image
        // compression level to use. 4 is the highest quality, but results in larger
        images.
    Image getGIF(LassoRequest request, String fieldName, int bitsPerPixel,
    boolean interlace)
        // Works identically to getJPEG(), but returns the image in GIF format.
        // bitsPerPixel can be from 1 through 8. Images can be optionally interlaced.
    String getLasso()
        // Returns the currently used Lasso name.
    LassoResponse processRequest(LassoRequest request) throws
IOException
        // Attempts to process the request defined by the request object.
        // Returns a LassoResponse object containing the resulting data.
class LassoResponse
    LassoResponse(String responseString)
        // Constructs a response given a response string.
        // This is used only by the LassoProxy class.
        // Do not call this method without a properly formatted response string.
    String[][] recordData(int index)
        // Returns all the fields and their values
    String[] fieldData(int recordNum, int fieldNum)
        // Returns the value for the specified field in the specified record
    String fieldValue(int recordNum, int fieldNum, int valueNum)
        // Returns a particular value from the specified field
    int numFound()
        // Returns the number of records which were found as a
        // result of the previously processed request
    int numRecords()
        // Returns the number of records which were retrieve as a result of
        // the previously processed request int numTotal()
        // Returns the total number of records in the targeted database
    int recordID(int index)
        // Returns the record ID of the record specified by index
```

```
        // Indexes are zero based
    int numFields()
        // Returns the number of fields on the target layout in the target database
    String[] fieldNames()
        // Returns an array of field names for the layout and database
    String fieldName(int index)
        // Returns the name of the specified field
    boolean fieldRepeats(int index)
        // Returns true if the specified field is a repeating field
        // false otherwise
    int repeatSize(int index)
        // Returns the maximum number of repeat values for the specified field
    boolean fieldNullsOK(int index)
        // Returns true if the field can be empty
        // false otherwise
    String[] fieldvalue_list(int index)
        // Returns the value list attached to the specified field
    int resultCode()
        // Returns the result code generated by Lasso when it processed the
request
        // Possible values and their explanations can be found in the
        // LassoResponse.java source file.
    int fieldIndex(String name) throws FieldNotFoundException
        // Returns the index of a field by name.
        // This index can be used to retrieve information about the field
        // in any of the methods which require a field index I.E.
fieldNullsOK()
    boolean hasvalue_list(String fieldName)
        // Returns true if the specified field has a value list
attached to it
        // false otherwise
    int fieldType(int index)
        // Returns the type of the specified field.
        // Possible values are:
        //  LassoRequest.BOOLEAN
        //  LassoRequest.CHAR
        //  LassoRequest.SHORT_INTEGER
        //  LassoRequest.INTEGER
        //  LassoRequest.SHORT_FLOAT
        //  LassoRequest.FLOAT
        //  LassoRequest.IMAGE
        //  LassoRequest.DATE_TIME
    String toString()
        // Returns the original response string used to create the
LassoResponse object
```

Appendix A: Tips and Techniques

The following section provides some tips and techniques for utilizing Lasso and improving Lasso and FileMaker Pro database integration solutions.

Use of Frames with Lasso

When creating HTML frame files, it is easiest to set up only one fixed frame to display a unified header for all pages, and then link from one frame directly to the next. It is possible, however, to link from one frame to another to complete a Lasso action. The trick is to specify the name of the target frame in the "form" tag. The format used would be (in this case with the Plug-in):

```
<form action="action.lasso" method="post" target="name-of-target-frame">
```

This tip was submitted by Rodney Capron, of Argent Media Group, Ltd. on June 12, 1997, and is used with his permission.

Here is a sample set of files that starts at the frameset file that indicates multiple frame sources to be loaded, in this case there are two frame sources:

```
<html>
    <head>
        <title>Our Homespun Site</title>
          </head>
    <frameset rows="110,*" border="1">
        <frame src="navbar.lasso" name="nav bar" scrolling="no"
noresize marginwidth="0" marginheight="0">
      <frame src="whatsnew.lasso" name="Main Content"
      marginwidth="10"
          marginheight="10"">
    </frameset>
    <noframes>
    <body>
      Viewing this page requires a browser capable of displaying frames.
  </body>
    </noframes>
</html>
```

The first page that is loaded into one of the specified frames is a "navigational bar" file. In this case an [inline: …] is used to reference the FileMaker database and show a form that has a value list. When the page is loaded, Lasso creates a pop-up selection list using the inline information, and this can then be used on the search form in which it appears. The "navigational bar" file is as follows:

```
<html>
<head>
<title>Our Homespun Site</title>
</head>
<body bgcolor="#ffffff" background="images/image.gif">
[inline:database="software_submissions",layout="whatsnew",display="yes",
      search,sortfield="submission date",sortorder="descending"]
   <center><table border="0" cellspacing="3" cellpadding="2"
   width="555">
<tr> <td colspan="4">
<font face="Arial">Welcome to the software archives, below are the most
recent additions and updates to our site. You can use our software category
selector in the top frame to go any category of the software
archives.</font></td>
</tr>
</table>
<table border="1" cellspacing="3" cellpadding="2" width="565">
<tr> <td bgcolor="#003194" colspan="5" width="547">
<b><font face="Arial" size="5" color="#ffffff">New Software And
Recent Updates!</font></b>
<font face="Arial" color="#ffffff">Click on the name field to see a detailed
record.</font>
j2<font face="Arial" color="#ffffff">There are [field: record number] software
packages available for download.</font>
</td>
</tr>
<tr height="29">
<th bgcolor="#c0c0ff" width="114" height="29">
<font face="Arial">Author Name</font> </th>
<th bgcolor="#c0c0ff" width="130" height="29">
<font face="Arial">Software Title/Version</font> </th>
<th bgcolor="#c0c0ff" width="69" height="29">
<font face="Arial">Submission Date</font></th>
<th width="90" height="29" bgcolor="#c0c0ff">
<font face="Arial">Software Category</font></th>
<th width="108" height="29" bgcolor="#c0c0ff">
<font face="Arial">Download File</font></th>
```

```
</tr> [record]
<tr> <td bgcolor="#e6e6e6" width="114">
<font face="Arial">[field: name]</font>
</td>
<td bgcolor="#e6e6e6" width="130"><font face="Arial">
<a href="[detail_link: layout=detail, format=detail format3.html]">
      [field: application title.version]</a></font></td>
<td bgcolor="#e6e6e6" width="69">
<font face="Arial">[field: submission date]</font></td>
<td width="90" bgcolor="#e6e6e6">
<font face="Arial">[field: software category]</font></td>
<td width="108" bgcolor="#e6e6e6"><font
face="Arial">
<a href="[field: url for zip file, raw]">Download Zip File</a></font> </td>
</tr>[/record]
</table>
</center>
   [/inline]
</body>
</html>
```

The first content page appears within a lower frame. When a product search is submitted, this page will be updated to display the product list. The file would show as follows:

```
<html>
   <head>
      <title>Our Homespun Site</title>
   </head>
   <body bgcolor="#ffffff">
      <center><table border="0" cellpadding="0" cellspacing="0"
      width="623">
<tr> <td width="301" rowspan="2">
<img src="images/logo.gif" align=top width="298" height="72"></td>
<td width="314" colspan="2" valign="top">
<img src="images/buttons.gif" width="303" height="48" border="0"
usemap="#buttons">
<map name="buttons">
<area coords="208,29,297,44" shape="rect" href="advertising.html"
target="Main Content">
<area coords="94,29,203,44" shape="rect" href="beta.html" target= "Main
Content">
<area coords="2,27,86,44" shape="rect" href="partner.html" target= "Main
Content">
<area coords="210,5,299,22" shape="rect" href="submit_news.html"
target="Main Content">
```

```
<area coords="96,5,203,24" shape="rect" href="submit_software.html"
target="Main Content">
<area coords="2,4,88,24" shape="rect" href="contactus.html" target= "Main
Content">
</map></td></tr>
<tr height="27"> <td width="127" height="27">
<img src="images/selector.gif" width="124" height="25"
align="absmiddle"></td>
<td width="185" height="27" valign="bottom">
   <form action="action.lasso" target="Main Content">
<input type="hidden" name="-database" value="software_submissions">
<input type="hidden" name="-layout" value="whatsnew">
<input type="hidden" name="-response" value="hitlist format2.html">
   [inline:database=software_submissions,layout=whatsnew, show]
<select name="software category" size="1"> [option: software category]
</select>[/inline]
</td></tr>
<tr>
<td colspan="3" width="617">
<img src="images/navbar.gif" width="615" height="17" border="0"
usemap="#navbar">

<map name="navbar">

<area shape="rect" coords="2, 1, 79, 16" href="whatsnew.lasso" target="Main
Content">
<area shape="rect" coords="85, 1, 169, 15" href="action.lasso? -
database=software_submissions&
-layout=detail&-response=software.html&-show" target="Main Content">
<area shape="rect" coords="175, 2, 259, 15" href="accessories.html"
target="Main Content">
<area shape="rect" coords="266, 1, 350, 15" href="http://www.us.faq.com"
target="_top">
<area shape="rect" coords="355, 1, 439, 15" href="related.html"
target="Main Content">
<area shape="rect" coords="446, 1, 528, 14"
href="https://www.flash.net:442/home/k/e/kenw/secure/secform1.htm"
target="Main Content">
<area shape="rect" coords="536, 1, 612, 14" href="news.lasso"
target="Main Content">
</map> </td>
</tr>
</table>
</center>
<input type="submit" name="-search" value="search for software">
</form>
</body>
</html>
```

From here, other pages can be set up to deliver a typical series of Lasso pages, to add, search, update, or delete records, and so on.

Hiding Lasso Format Files Using Frames

HTML "frames" are also useful for hiding the HTML source of returned format files by restricting the HTML of the format file to the target frames. This prevents someone from being able to create a URL to display the raw HTML page in order to view field, database, or file location information. To make these pages even more secure, have the first accessed page processed by Lasso before the page is returned to the client browser. This is accomplished by using Lasso as an action through various methods, see SUFFIX MAPPING in CHAPTER 6: LASSO SERVER for more details.

Submitting the Form Without a Lasso Action

Do not specify database activity with the form on the search page. Rather, design this form as a means to capture information from the user and "pass" it to the next page. This can be accomplished by not specifying a Lasso -database, -layout, or -search action. Instead, remove all of your hidden fields except the -response field and change every occurrence of -search to -nothing.

Using Frames to Pass Arguments

The following page is a frameset. One of the frames specifies a request to Lasso such as:

```
<frame src="action.lasso?[database]=example.fp3..." name="hitlist">
```

Build into the <frame src> an embedded URL (see CHAPTER 4: LASSO METHODOLOGY). Recapture the original submitted data with the [form_param: …] tag. As an example, one "name=value" pair within the URL could be:

```
...&Name=[form_param:Name]&...
```

Lasso 2.0 and earlier cannot retrieve the value of operator, so to allow a user to select an operator for a field, rename the operator-field from [op] to some unique name such as "Nameop," then retrieve that value in the frameset with:

```
...&-operator=[form_param:Nameop]&...
```

The embedded URL will end with the search action:

```
...&-search" name="hitlist">
```

Testing Without a Network Connection

When developing Lasso format files it is imperative to have a Mac OS Web server and a FileMaker Pro database. This doesn't necessarily mean an Internet connection is required to test the format files. To test or design a Lasso solution without an Internet network connection, an "internal" Internet can be created. A Mac OS Web server is needed, but there are several options for free or inexpensive Web servers to use for development purposes.

To set up the testing environment without a network:

Using Open Transport

1. Open the TCP/IP control panel found in the Control Panels folder/ System Folder.
2. Under "File" on the menu, select "Configurations."
3. The current settings are already saved so it isn't necessary to save them.
4. Duplicate the settings and then rename, e.g., "no network."
5. Make the new settings active, by selecting "Make Active."
6. Set up the new configuration:
    - Connect Via: AppleTalk (MacIP)
    - Configure: Manually
    - IP Address: Any IP, including the actual one, for example, "10.10.10.10" since it is easy to remember.
    - MacIP server zone: <Current AppleTalk Zone>
    - Ignore other settings.
7. Close TCP/IP. When the control panel is closed the "no network" settings are saved.
8. Launch your Web server application. If it is already open, restart your Web server.
9. Launch your Web browser and enter the URL as:

    http://10.10.10.10/filename.html

After testing the format files in the no-network environment, the previous settings can be restored by opening the TCP/IP control panel and making the original "Default" settings active. To restore, do the same for the no-network settings. With Open Transport, use a "hosts" file to map the "10.10.10.10" address to some domain name.

Optimizing Performance

The response time of Lasso depends on many factors that are specific to each Lasso solution. Nevertheless, there are some steps that can be taken to optimize the return of data from the FileMaker Pro database. The following guidelines will assist with understanding how to maintain top performance.

Database as Single-User — The best access to data in a FileMaker Pro database is when that database is open in single-user mode on the Web server, and located on a drive that is directly attached to the Web server (not from a shared volume).

Limiting Fields on Layouts — Specifying all of the format files to use a single layout that contains all of the fields which might possibly be used is an inefficient way to set up the FileMaker Pro Web database and is not recommended. When searching the database, Lasso retrieves all of the data from all of the fields in the found set on the specified layout, regardless of whether or not the fields are referenced in the format file (this is due to the way FileMaker Pro makes this information available).

Create Layouts Specific to the Action — If the fields on the layout are limited to only those used by the format file, Lasso performs more efficiently. For an even greater boost in performance, create layouts that contain only fields pertinent to the Lasso action.

Sorts — On large databases, sorting can dramatically affect performance. This is a FileMaker Pro imposed limitation. If sluggish performance is reported with a Lasso solution, it may be due to lengthy sorts. Make sure to manually index all fields used by Lasso for searching and sorting. If using calculation fields, periodically re-index the fields as records are added, perhaps through a "maintenance script."

Contains Searching — On large databases, performing "contains" type searching can dramatically impact performance. This is a FileMaker Pro imposed limitation due to the way data is indexed. Other search operators do not affect performance when searching databases of any size.

Encoding Large Text Fields — Character encoding can impose performance hits with large text fields. Thus, if possible, return fields as raw. For example: Our tests show display of 64k (FileMaker Pro maximum limit) of text in15 seconds with character encoding on (default and/or "URL" type options) and at 1 second with it off (set to "raw"). Unless the features of encoded text are necessary, set large text fields to "raw" for optimum performance. Keep in mind that HTML encoding of special characters will result in those characters not being displayed properly by the browser. Fields that were created from selection lists can always be returned "raw" since their values will be predetermined (that is, if no special characters are used as value list items).

Limiting Records Returned — Limiting the number of records returned (by setting the -maxRecords value) can increase the performance of the Lasso/FileMaker Pro Web database solution. Users can retrieve additional sets of records by using the [next] and [prev] commands.

Reduce the Number of Calculation Fields Used — Make sure all calculations used for searching are stored and not calculated when needed. If a lot of records are being added, fields need to be re-indexed periodically. If the database is closed they should be re-indexed when re-opened. If a database has numerous records, the timed difference between indexed and non-indexed fields will be apparent after the database is re-opened.

Conditional Statements — Using conditional statements within something that loops (record, portal, repeating, loop or while) will slow Lasso response time.

Serving Pages from Other Platforms

Lasso cannot process format files stored on other servers. However, if using the strategy of serving an entire Web site from a server on another platform (e.g., Windows NT or UNIX), create a static HTML form and place it on that server. When the form is submitted it would be processed by the Mac server if the form action is specified as follows:

```
<form action="http://Mac-server.domain/Lasso.acgi" method="post">
```

The Mac OS server address is denoted by "Mac-server.domain." The response needs to be a page served by the Mac and is not possible to transfer data for display on the UNIX or NT page. Of course, in this scenario Lasso can't be used to auto-populate value lists on the add form, or be able to do updates to a record, unless the updating was done on the Mac server.

FileMaker Pro Server

Lasso can work with a single-user database, as a host of a multi-user database, or as a guest to a multi-user database. In all cases, the client version of FileMaker Pro needs to be open on the Web server, with the database served by Lasso open as either "host" or "guest." If the database to be used with Lasso needs to also be available on a local network, FileMaker Pro Server is the best option for sharing the FileMaker Pro databases. FileMaker Pro Server speeds the access to the database and is much preferred to sharing the database by having it open as "multi-user" and shared directly from another "client" version of FileMaker Pro.

FileMaker Pro Server serves as the host of a shared database and speeds its network functions by making the database multi-threaded and supplementing the performance of CPU intensive tasks such as sorting. FileMaker Pro Server understands only a very few Apple Events, so it doesn't allows Lasso (or any other product) to interact with it directly. The database must still be opened by the client version of FileMaker Pro, even if this has to be on the same computer.

FileMaker Pro Server can either be run on a dedicated computer or on the same computer as the Mac OS Web server application. Lasso, FileMaker Pro, FileMaker Pro Server, and the Web server software can all exist on the same machine. As one might expect, the performance of all applications will be diminished when competing for CPU resources. Also, by having the Lasso-served database open as a "guest" further diminishes Lasso's performance.

Alternatively, the Web server and FileMaker Pro Server can run on separate computers on the same network. The FileMaker Pro Server has the served database open as "host." The second computer would house the Web server software and a copy of FileMaker Pro ("client") with the Lasso served databases opened as "guest." This may also slow the server due to network slowdowns as data is transferred between host and guest.

The best situation is to have the FileMaker Pro database opened as single-user with no guests on the same machine as the Web server and Lasso. If this is not an option for a particular situation, analyze the solution's particular needs to determine which setup is best. There are many factors that affect this decision, but all usually involve a trade-off between the speed of the Web served database and the availability of that data through a local area network.

Multiple Servers

Most Lasso integrated database solutions can easily be served on a single Mac OS Web server. In situations where it is better to focus all database activity on one server, Blue World highly recommends the Lasso Server version. However, if the load on the Web database is still too high, try to spread Lasso and the database over multiple servers. How this is accomplished depends on whether the database contains a dataset that must be kept together (i.e., a contacts database), or if it can be duplicated into identical copies (e.g., a catalog).

If the solution doesn't depend on having all of the records in the databases together in the same database, then have the DNS server share the load by switching round-robin to multiple identical servers (with a copy of your FileMaker database open on each). In other words, the DNS server would route a new visitor to the next server in the sequence. Of course, if the structure of one database was updated this would also have to be done for all databases. This can be handled by having one master database with only sample records to make changes to. When it comes time to update the databases, create a clone of the master and import the data from the other servers. On the other hand, if collecting information, the data could be collected on the separate servers and later imported to one final source database.

Alternatively, if the data must stay together and be made available as one dataset, a FileMaker Pro Server could be utilized to share the main database over the network. Each Web server would need to open a copy of the database as "guest" with FileMaker Pro ("client"). The DNS server still needs to share the load by switching the various servers in a round robin fashion. Refer to OPTIMIZING PERFORMANCE in APPENDIX A: TIPS AND TECHNIQUES for details on more efficiently utilizing the database to minimize the impact of networking the database.

As mentioned in the previous section in regards FileMaker Pro Server, shared access to the database is slowed due to network latency as data is transferred between host and guest. Nevertheless, if the situation requires it, this approach makes it possible to network the database to multiple servers to maintain a dataset. The trade-offs need to be weighed for every specific situation. It just might be that the best option is to use one computer to carry out all functions, and maximize the ability of the Web serving computer to handle the load by purchasing more RAM, improving your bandwidth, getting a computer with a higher MHz rating and so on.

Using Lasso as the Default Action for All Pages

If Lasso is set to process all files that end with the extension ".html" then the Lasso application will serve as a pre-processor. That is, Lasso will check every client request before it gets to the Web server application. This would be useful for setting cookies, recording the referring URL, running conditional statements, etc.

To set up Lasso as an action for all files ending in ".html," or any other suffix, configure the Web server application to process these files with the Lasso application. For example, with WebSTAR an action is created using the "WebSTAR Admin" application. To accomplish this, launch the WebSTAR Admin application, then under the "Configure" menu, open "Suffix Mapping." Find the suffix ".html" and set the action to "LASSO_PLUGIN" (or to the Lasso.acgi). Set the type, creator, and MIME type to "*" and click the "REPLACE" then "UPDATE" buttons. For more information about setting an action see SUFFIX MAPPING in CHAPTER 6: LASSO SERVER.

In the HTML, wrap each page in a conditional. That is, surround all of the HTML on the page within an [if:] statement.

For example:

```
[if: cookie:cookie_name=blah]
   [inline: …use this inline to add a record for each user …]
   <html>…
   Your normal html page goes here.
      …</html>
   [else]
      [include: "/path_to_file/form.html"]
   [/if]
```

The [if: …] should force them to fill out your form and the [inline: …] will log their activity to a database. It is possible then to log each request to one database and then view those records through a portal in a database related by the user's IP address.

Creating Sub-Summary Headers Within a Search

When outputting search results, a sub-summary header is sometimes needed to categorize the returned information in a report format. The following describes how to create subsummary headers after completing a Lasso search.

An example of search results with subsummary headers would be:

Human Resources Department

Jane Doe
Bob Smith
John Doe

Accounting Department

Mike Alpha
John Beta
Lucy Ceta

Legal Department

Allen Delta
Jose Orteca

Two related databases are needed to provide search results with subsummary headers. In the above example, the main database, called departments, would contain records with the employee name and their department. The supporting, related, database would contain records of the departments. set up a relationship in the department database to relate it to the employee database by the department field. For this example, the relationship name will be "Employee Link." Create a layout in the department database that contains a portal showing the employees listed in that department along with any other information to show for each employee in the list. This layout should contain all of the fields involved in the search or the returned results.

Create a search page to search the departments database. The database name specified on the search page for -database will be the departments database, and -layout will be the layout with the portal. The portal containing the related records can be searched, but make sure the field being searched is within the portal and that the related field is specified by using "relationship::fieldname." Make sure the portal is given a scroll bar on the layout. Check that the search is possible by doing a find directly in the portal found in the departments database. Keep in mind that only related values may be searched and searching cannot occur on all of the records in the related database.

Create the results page using Lasso's portal code. For the above example, the Lasso code in the results page would look like this:

```
[record]
<h3>[field: "department"]
</h3> <dl> [portal: "Employee Link"]
```

```
<dd> [field: "Employee Link::name"]
[/portal]
</dl>
[/record]
```

"Department" is the name of a field in the departments database. Employee Link is the name of the relation from the departments database to the employee database. "Name" is the name of a field in the employee database. It is shown on a portal within the layout specified in the search document in the department database. See CHAPTER 10: REPEATING FIELDS AND RELATED FIELDS for more information about using Lasso to display values within portals.

This tip was submitted by Diana Oswald (diana@acmetech.com) and is used with her permission.

Another option for creating a sub-summary display is to use the [inline: …] tag.

Backing up a FileMaker Pro Database

There are different methods for remotely backing up a database without downloading the entire actual database. Here are a few suggestions.

First create a format file called "File.txt" formatted as follows:

```
[record][field: "field1"],[field: "field2"],[field: "field3"]

[/record]
```

To include any fields that should be returned, specify every field or just a few important ones. No HTML codes are necessary on this file.

Use the following link to have all (or some) fields returned as comma separated text:

```
<a href="action.lasso?-database=database&-layout=search&
-response=file.txt&-findall">DownloadDatabase</a>
```

To download the database, hold the mouse button on this link and choose "save as" to avoid loading in the browser window. If this particular database is protected by Lasso Security, you'll be prompted for a username and password. The data will be returned with each field separated by a comma and a return placed at the end of each record.

This tip was submitted by Mark Sassman (asap@iquest.net) on May 14, 1997 and is used with his permission.

Another method is to create a script in FileMaker Pro that finds all records and saves a copy of the database to a specified location. The files can then be transferred via FTP from that location. The database doesn't need to be closed to accomplish this, but the Script may bring FileMaker Pro to the front, so make sure the Web Server is returned to the front as soon as possible by using an AppleScript, as described in CHAPTER 11: EXECUTING FILEMAKER PRO SCRIPTS.

Select Field to Search on from Pop-up List

It is possible with Lasso to first type in a value into a text entry field, pick the field to search on from a selection list, and then have the search occur on that selected field. The following tip uses the [form_param: …] tag to return values to an inline, which is where the actual search occurs. To do so, first create a "search" format file, which is used to gather the field and values to use on the actual search that automatically occurs on the following format file. The action in this case is -nothing since you don't actually want to interact with the database at this point:

```
<form action="action.lasso?-response=filename&-nothing" method="post">
```

The database and layout do not need to be specified. However, if "filename" is the file to be used for the response, you should indicate the relative path to the file.

Then, add an input (there can be several of these):

```
<select name="Searchfieldname">
   <option>Field1
   <option>Field2
</select>
<input type="text" name="SearchValue">
```

A submit button is then created to allow the user to initiate the "search":

```
<input type="submit" name="-nothing" value="Search">
```

In the above code:

"Searchfieldname" — Is any text, it does not matter what the name is, it is a temporary holder for the name of the field.

"SearchValue" — Again, this is any text, it does not matter what the name is, it is a temporary holder for the value that you want to use in the real search.

"Field1" — Is the actual real name of the field that is in your FileMaker Pro data; the same applies to "Field2," and there can be as many as you would like.

The search actually occurs on the next format file (specified by the -response tag on the first file). This format file contains an inline tag that would appear as:

```
[inline: database="dbname", layout="layoutname",
   form_param:"SearchFieldname"=form_param:"SearchValue", search]

   The field to display for the result is: [form_param:"SearchFieldname"]

[/inline]
```

The "[form_param: …] tag simply retrieves the values entered into the previous form, thus enabling them to be used again in the search that is processed in the inline on the response format file. In other words, the [form_param: …] tag is used within the inline to return the search parameters from the last search, in this case "Searchfieldname" (the name of the field to search in, which was selected from the previous pop-up list), and "SearchValue" (the value to be searched, which was entered temporarily into a text entry field). The "dbname" and "layoutname" are the actual names for your database and layout.

The result appears since the field with the result (found within the [inline] … [/inline] container) is also identified as being the same field the search is occurring on, that is, the value entered into "Searchfieldname." Several fields can be displayed in a similar manner. Note that this process can be extended for use with other types of selection lists such as checkboxes.

Miscellaneous Tips

Case Sensitivity

In general, case doesn't matter when dealing with database or layout names. But, case does matter when specifying values from value lists. Selection list options must be spelled exactly, with the same capitalization, on the format file as they appear in the FileMaker Pro database. As a rule, make sure the case of values specified in Lasso format files matches that found in the FileMaker

Pro value list. This is not an issue, however, when auto-populating value lists using the "post-Lasso" method and is one big advantage of utilizing this feature.

Auto Refresh With New Data

With Lasso, field data may be returned on an "add," "update," or "delete" reply page. In addition, it is possible to automatically link to another format file by using the [header] tag.

Value Range Searching

There are two ways to search for field values that lie within a range of values.

1. Create two inputs for the same field in your HTML search format file. Add a search operator to look for values greater than, or after, the first field, then add a second search operator for the less than, or before, operator. Values that are equal to the value requested may also be included.

```
Date<select name="-operator">
        <option value="gte"> from
        <option value="gt"> after
    </select>
    <input type="text" name="Date" size=10>
        <select name="-operator">
        <option value="lte"> to
      <option value="lt"> before
</select>
<input type="text" name="Date" size=10>
```

Optionally, the operators for "less than" or "greater than" can be hard-coded by using the "hidden" field parameter.

```
Date<input type=hidden name="-operator" value="gte">
    <input type="text" name="Date" size=10>
        <input type=hidden name="-operator" value="lte">
    <input type="text" name="Date" size=10>
```

2. The second option is to separate the range values with three periods "…" in the form:

```
xx/xx/xx...xx/xx/xx
```

The symbol "…" (three periods) is the same optional operator for range searching that is used with the find command in FileMaker Pro. This second approach may only work with date range searching.

Value range searches will not work with the default "begins with" operator, but requires the use of the "gte" operator as follows:

```
<input type="hidden" name=-operator value="gte">
```

Tips on Rebooting After a Crash

Although Mac OS Web servers are reliable, no Web server is perfect. If perchance the system crashes, upon reboot, make sure to refuse connections to the Web server (set within the Web server) while FileMaker reconstructs the databases. Unfortunately, this can be a lengthy process, depending upon the size of the databases. By refusing connections after a reboot, this will ensure that Lasso.acgi is not called BEFORE the Lasso Security databases have a chance to relaunch. In addition, if using the Lasso Server or the Plug-in version, and using Lasso Security, precautions should be taken to not start the server until the Lasso Security databases have had a chance to open.

Open Shared Network Database on Web Server

The following is a description of how to open a shared or networked database that uses password restrictions without encountering a dialog. This technique is useful when the database needs to be opened automatically on a server so it is available to Lasso after a server crash. In this case, the shared FileMaker database is protected with passwords for certain users and the master password cannot be automatically entered. When the server is restarted the password needs to be entered manually when the database opens.

Here are several options for accomplishing this:

1. Set up a limited access password for the database that allows browse, print, and export; and if adds are allowed, then also include create records. Have this password auto-entered when the database is opened by modifying the Document Preferences to "Try Default password."
2. Have a "dummy" database reside on the server and open the shared database over the network via a script. Here is how to accomplish this second option:
    a. Create the dummy database with one or no fields (none are needed unless you want this to be a log that records the date and time of the restart by auto-entering these values into a new record).

b. Create a script, i.e., "Open Network Database" with the following two steps:

   Step 1 — "Open," select the shared database using Specify File" and then click on Hosts button to locate the shared database on the network

   Step 2 — "Close" (nothing selected, this is to close the dummy database)

c. Use the Access Privileges/ Define Passwords command to create a password. This must be the SAME PASSWORD as the master password in the shared network database.

d. Open Preferences and then go to the pull down for "Document" select "Try Default password" and enter the password created above.

e. Select "Perform Script" and then enter the name of the script, i.e., "Open Network Database"

f. Place this file in the startup items of the Mac OS Web server computer.

If the file is shared using FileMaker Pro Server on the same computer (that is the Mac OS Web server computer) then add a time delay of 30 seconds in the "Open Network Database" script using the "Pause/Resume Script/ and the "Specify" button to set the duration of the pause.

As long as the master passwords are the same in each one, use this dummy database to open several networked databases with the same script.

3. A third option was submitted by Eric Westlund (ewest@hatcher.com) on the Lasso Talk mailing list. He suggested:

   "Write a small AppleScript that launches FileMaker Pro and opens the databases using a limited password on startup such as…

```
tell application "FileMaker Pro"
    activate
    open alias "hd:Web DBs:guestbook" with password "password"
    open alias "hd:Web DBs:Lasso Security" with password ""
end tell
```

Note: Unfortunately, this technique won't help if the database was damaged after the crash and needs to be recovered, though this could be added with some additional FileMaker scripting.

Addendum

See APPENDIX E: SUPPORT to find additional sources of Lasso tips.

Appendix B: Character Sets and Translation

The ISO Latin-1 and Mac OS Character Sets

The ASCII character set defines a universal set of characters for all Roman-based languages by assigning characters a numerical code. The most basic set of characters that are recognized by all computer systems have been assigned a decimal value from 0 to 127.

The Mac OS uses a standard character set known as "Standard Roman," while the Internet uses "ISO Latin-1" (for Western Europe) for the HTML character set. Both Mac Standard Roman and ISO Latin-1 use the ASCII character definitions for characters in the range 0 to 127, and so these characters are the same in both.

Mac Standard Roman and ISO Latin-1 use different character sets for characters above 127. Generally, these special characters are in the range 128-255 in the ASCII character set. Some of these are the same, and some characters defined in one set are not defined in the other.

HTTP and HTML Character Encoding

The HTML and HTTP Internet standards require that certain special characters and all non-ASCII extended characters be encoded. The non-ASCII extended characters are those with a numerical value in the range 128-255. Character encoding is the process of replacing an extended character in an HTML document with a "character reference" or "entity reference."

The HTML standard reserves certain characters that must be encoded if included in an HTML document. The special characters are right angle bracket (>), left angle bracket (<), quote ("), and ampersand (&). The following character reference is used to display these special characters:

```
&#nnn;
```

In this format, "nnn" is the numerical code for the character.

The HTTP standard requires that all reserved characters with special meaning, as well as certain "unsafe" characters, be encoded if they are to be used in an Internet URL or email address. The reserved characters are:

; / ? : @ = &

The unsafe characters are:

> < # % { } ‘ | \ ^ ~ [ ] ` ¨ © "

The following character reference is used to display these special characters:

%nn

In this format, "nn" is the hexadecimal value that represents the character in the ISO Latin-1 scheme.

Lasso Character Translation

Lasso correctly displays many of the special characters used by Macintosh computers by performing translation between the ISO Latin-1 and the Macintosh Standard Roman numeric representations. In other words, Lasso performs HTML encoding of all values from the FileMaker Pro database when substituting values in an HTML format file. For example, when displaying the contents of a field using the [field: …] tag, Lasso does **not** translate or encode characters in the HTML portion of your Lasso format files (that is, everything that is not returned by Lasso from FileMaker Pro).

Mac Standard Roman characters that are not defined in ISO Latin-1 (for example, the bullet generated by option-8) are translated to an asterisk. The exception is smart quotes on the Mac (which doesn't exist in ISO Latin-1), which Lasso translates to regular quotes.

Any Standard Roman characters that appear within FileMaker Pro database fields are automatically translated and encoded by Lasso when data is returned to a Web browser. Thus, any language that uses the Standard Roman character set will be displayed correctly by the Web browser. The following languages use the standard Roman character set:

English	French	German
Italian	Dutch	Swedish
Spanish	Danish	Portuguese
Norwegian	Finnish	Icelandic
Maltese	Turkish	Lithuanian
Estonian	Latvian	Croatian

There are character sets that this release of Lasso will not translate properly, since they define a different set of characters for the ASCII values 128-255; for example, the Macintosh character sets used by Central European and Cyrillic languages. There are also other HTML character sets besides ISO Latin-1 (such as ISO Latin-2) that contain characters Lasso cannot translate.

Not all versions of Lasso support languages that use two-byte character sets, as do Japanese, Chinese, and Korean (Roman languages use one-byte character sets). A separate version of Lasso that is specifically designed for 2-byte languages is required. Information regarding versions of Lasso that work with two-byte character sets is available at the Lasso Web site.

Field Encoding

Exchanging information between the world's multitude of languages and computer systems presents a translation challenge. The Macintosh uses the "Macintosh Standard Roman," character set, while the World Wide Web's HTML uses ISO Latin-1. Lasso, therefore, must translate (or "encode") to the correct language for each medium. Lasso, by default, HTML-encodes field values. For example, "<" will be encoded as a "<" string.

Disabling Character Encoding

If your FileMaker Pro field data includes HTML tags, you'll want to prevent certain characters from being encoded. In addition, including field data as part of a URL requires additional efforts to prevent other characters from being encoded. Because of this, several keywords can be used by Lasso substitution tags to control how the data from FileMaker Pro fields is encoded. These keywords include "raw," "url," break," and "smart."

The keywords are applied in the following manner:

```
[field:"Field Name",break]
```

Or:

```
[token_value,url]
```

Or:

```
[field:"Relationship Name::Field Name",smart]
```

Keywords are used with repeating fields slightly differently, as follows:

```
[repeating:"FieldName"] [repeat_value,url] [/repeating]
```

Note that "url" cannot be used with [repeating: …].

You cannot specify multiple "smart," "url," "break," or "raw" keywords for the same field. A space between the colon and field name is ignored.

Although some browsers will recognize extended characters properly in their raw form, others require extended characters to be encoded according to the HTML character encoding scheme. Similarly, with extended characters in URLs it is important that these special characters are encoded so that the Web server can direct the request properly. If you use raw values, be aware that some Web browsers may have problems displaying the characters properly.

Using the "raw" keyword will speed up Lasso's performance, particularly with very large text fields. Character encoding can dramatically affect performance on large text fields. For example, our tests showed display of 64k of text in around 15 seconds with character encoding on (default and "URL" type), and in 1 second with it off (set to "raw"). 64k is also the FileMaker Pro maximum limit for any given field.

Using the Raw Keyword to Disable Character Encoding

The HTML standard reserves certain characters that must be encoded if included in an HTML document. When the "raw" keyword is used with a Lasso field command, characters are not altered (e.g., encoded) when Lasso returns field values from the FileMaker Pro database.

To ensure that special characters in field values are interpreted as HTML tags, turn off HTML character encoding for the specified field. The field data will then return exactly as it exists in the FileMaker Pro database field. This is accomplished by adding the "raw" keyword to the [field: …] tag:

```
[field:"FieldName",raw]
```

However, if HTML coding is turned off in this manner, extended non-ASCII characters won't be encoded. This may produce undesirable results depending on the browser. Therefore, avoid using extended characters in fields which will have HTML character encoding turned off.

Using the URL Keyword to Encode with the HTTP Standard

When the "url" keyword is used with a Lasso field command, characters are encoded according to the HTTP standard, rather then the HTML standard. This is accomplished by adding the "url" keyword to the [field: …] tag:

```
<img src="[field:"FieldName", url]">
```

If you are going to use field values in URLs (for example, inside an anchor or image tag), instruct Lasso to perform URL encoding rather that HTML encoding. For example, if a field value is returned with an embedded URL, use the "url" keyword to encode spaces or HTTP reserved characters (unless these will not be present, such as in a field that is defined to be a "number" field in FileMaker Pro). The reserved characters are:

```
; / ? : @ = &
```

The unsafe characters are:

```
> < # % { } ‘ | \ ^ ~ [ ] ` ¨ © "
```

Using the Break Keyword to Include Returns

The "break" keyword can be used with the [field: …] tag to translate all carriage return characters to HTML
 tags. It is used in the format:

```
[field:"FieldName",break]
```

Lasso performs HTML character encoding (default behavior) when the "break" keyword is specified. The "break" keyword cannot be used in conjunction with the "raw" or "url" keywords to the [field: …] tag.

Using the Smart Keyword for Extended Character Encoding

Smart encoding will encode non-ASCII or special character entities, but will not convert the characters "<" and ">." With the "smart" keyword, special characters can transfer from FileMaker Pro database fields along with characters used for HTML coding. Thus, dynamic HTML can be utilized on the returned format file, along with extended non-ASCII characters. The "smart" keyword is used in a manner similar to "url," "break," or "raw." For example:

```
[field:"FieldName",smart]
```

Global Encoding and Decoding

Unlike field encoding, global encoding and decoding applies to literal text strings, substitution tags and sub-container tags. The following tags are components of the "Encoding_Tags.mod" module. They can contain multiple parameters, such as literal strings or substitution tags. Each parameter is surrounded by quotes and separated by commas.

The [encode_url: …] substitution tag concatenates all specified text together and then URL-encodes that text string.

For example, the following code:

```
[encode_url: "this is some text ", "some more text"]
```

Outputs the following result:

```
this%20is%20some%20text%20some%20more%20text
```

Lasso substitution tags can also be included. For example:

```
[encode_url: "http://", client_ip, field:file_path]
```

Outputs the following result:

```
http%3a%2f%2f202.252.25.63employees%2fdetail.html
```

The [decode_url: …] tag concatenates all specified text together and removes any URL encoding from the text string. This tag can be used to reverse the effect of the [encode_url: …] tag.

For example, the following code:

```
[decode_url: This%20is%20%3cb%3eth%e9%20string%3c%2fb%3e%d]
```

Outputs the following result:

```
This is <b>thé string</b>
```

The [encode_breaks: …] substitution tag concatenates all specified text together and then applies "Break" encoding. Break encoding translates all carriage return characters into HTML
 tags.

For example, the following code:

```
[encode_breaks: "This is", " ", "<b>thé string</b>
2 B encoded..."]
```

Outputs the following result:

```
This is &#60;b&#62;th&#233; string&#60;/b&#62;<br> 2 B encoded...
```

The [encode_raw: …] substitution tag concatenates all specified text together and then applies "Raw" encoding. Raw encoding

translates the Mac standard character set to ISO Latin-1, but does not convert HTML entities.

For example, the following code:

```
[encode_raw: "This is", " ", "<b>thé string</b>"]
```

Outputs the following result:

```
This is <b>thé string</b>
```

The [encode_html: …] substitution tag concatenates all specified text together and then HTML encodes that text string. HTML encoding translates all extended characters, including angle brackets, into their HTML entities.

For example, the following code:

```
[encode_html: "This is", " ", "<b>thé string</b> 2 B encoded…"
```

Outputs the following result:

```
This is &#60;b&#62;th&#233; string&#60;/b&#62; 2 B encoded…
```

The [encode_smart: …] substitution tag concatenates all specified text together and then applies "Smart" encoding. Smart encoding encodes extended characters into their HTML entities, but does not convert angled brackets, allowing embedded HTML to pass through.

For example, the following code:

```
[encode_smart: "This is", " ", "<b>thé string</b> 2 B encoded…"]
```

Outputs the following result:

```
This is <b>th&#233; string</b> 2 B encoded…
```

Appendix C: Troubleshooting

Refer to these guidelines when confronted with problems using Lasso. If you still have difficulties after reviewing these guidelines, please follow the procedures for receiving technical support. Oftentimes, the answer to the problem is found by searching the archives of the Lasso Talk email discussion forum.

When troubleshooting a Lasso integration project, it becomes extremely helpful to look not only at the HTML of the format files in their raw form, but also to view the source of the file returned by Lasso. To do so, use the "View Source" command in the Web browser to see how the file has changed after Lasso has processed it.

Receiving "No records found" Message

The best approach when you come across this type of situation is to make a copy of your add form and either delete or comment out all fields except for one. This one field should be a plain text field that has no restrictions, validations, or auto-entered values. Next go to the FileMaker Pro layout your format file is referencing and make sure this field is there with "allow entry into field" checked in the Field Format dialog. Make sure to include the basic elements for an add form:

```
<form action="action.lasso?-add" method="post">
<input type="hidden" name="-database" value="YourDBName">
<input type="hidden" name="-layout" value="YourLayoutName">
<input type="hidden" name="-response" value="insertpath/Add_Reply.html">
<input type="text" size=30 name="InsertYourFieldName">
<p><input type="Submit" name="-add" value="Add Record">
<input type="hidden" name="" value="">
</form>
```

Test to see if you can add one record and that the field value was submitted. If it was not, check the database, field, layout, and response names to make sure they are exact. If a field has any extended characters such as slashes or periods in the name, change it to something simpler. Make sure that the database and layout are not the same.

Next, add another field that is a text field and check it for restrictions as with the first. Test the add form again by entering data for the two fields. Check to see that a record was added with the two fields completed.

Add a third field and test. Keep repeating this process until you discover the field that is not being entered. Then check that field out on the FileMaker database. Retype the name in Define Fields and where it is specified on your format file. By following this procedure, I find the problem more times than not.

If the form you are checking is a search form you can follow a similar process as is described above. For the search, also check if a field is defined as a date or time type field. If so, then an operator must be specified as either "equals," "greater than," or "less than," the default operator "begins with" cannot be used with date or time fields. In addition, check the sortfield tag and make sure that it specify valid field names. The field you are sorting on must be defined in the database, though it does not need to be on the layout specified for the action. Other search parameters are OK since they use a default value if an incorrect value is specified.

Lasso Does Not Seem to Operate At All

If Lasso quits unexpectedly upon launching and does not get as far as the serial number/test drive dialog, first check that the required System extensions have been installed. Consult CHAPTER 2: INTRODUCTION for specific information on these extensions. Also, try to allocate more memory to Lasso.

- If more memory does not solve the problem, make sure the Mac OS Web server is running and check that all referenced FileMaker Pro databases were opened before Lasso was launched. All databases should be open with a "master password," so that there are no restrictions to entering data into a field. Make sure the format files are being opened via the server, not directly from the hard drive (e.g., use http://your-server-name/path-to-file/file.html)
- Confirm that the correct version of Lasso is specified in the form action. And, if using the CGI, check that the path location to Lasso is correct. For more details, see CHAPTER 4: LASSO METHODOLOGY.
- Confirm that the path to the various format files is correct, as discussed in CHAPTER 4: LASSO METHODOLOGY.

Nothing is Added to the FileMaker Pro Database

If attempting to complete an add or update and Lasso is unable to complete the action, check the following:

- If the "Lasso_Security.fp3" database was open before Lasso was launched (or if, using the Plug-in, it was opened before the Web server was launched), then Lasso Security is enabled. Close the security databases and restart Lasso with security disabled. Then, try the files again to see if the problem is related to a security issue.
- If you are getting an error message, note the error number and message displayed. Some error messages are generic; however, others could provide some specific information that will help resolve the problem. Refer to APPENDIX D: ERROR CODES for a list of the known error codes and suggestions for correcting the problem for each error.
- Confirm that the layout specified in your Lasso format file is available in the FileMaker Pro database and that all of the fields in the Lasso format files (both the form and the reply) fulfill the requirements discussed for each action below.
- **Add —** For adds, the most important recommendation, and the one to try first, is to open the FileMaker Pro database and switch to browse mode. Go to the layout specified in the format file. Try to create a new record and then add data to each field using the FileMaker Pro interface directly. Click outside of the fields to save the record. Try adding the same data that was added via the Web form. The FileMaker Pro database layout must contain all the fields specified in the "add" format file, **AND** all fields specified in the format file specified by the -response command tag. If a field is specified that is not on the layout, the "add" will occur but the data that was entered for the missing field is lost.
- **Update —** Troubleshoot in a manner similar to "add" routines. Also, check that the record ID has been specified in the form, in addition to the database, layout, and response file. The -recid is indicated on your HTML form with the following tags:

  ```
  <input type="hidden" name="-recid" value="[recid_value]">
  ```

  • Open the FileMaker Pro database and switch to layout mode. Select "Field Format" (from the "Format" menu), and

confirm that the option is checked to "Allow entry into field" (found at the bottom in the "Behavior" section). Make sure each is formatted according to its expected behavior. For example, fields using selection lists should be associated with a value list.

- While still in layout mode, make sure that there are no "undefined" fields in the layout. These would show as fields that are not assigned to any specific field and are blank where normally one would see the field name. An undefined field can accidentally be created if a layout is copied from one database and pasted into a database which doesn't have the same field definitions.

- Check that all database, layout, file, field, and other names or files are spelled correctly. Try to remove occurrences of any extended or reserved characters from these names (i.e., slashes, under scores, ampersands and so on, as is mentioned in APPENDIX B: CHARACTER SETS AND TRANSLATION). Although Lasso is not case-sensitive for many of these elements, try to copy them following the capitalization used. Drag names from FM Link for accuracy. Confirm that there are no trailing or stray spaces, or null characters (that is, ASCII character 0 [zero], which is sometimes created when pasting info copied from another application). Retype the entire file names in the Finder or in the define fields dialog box, or at least retype the first or last letter by dragging the letter to one side or the other and retyping that letter. As a last resort, try changing the name to something entirely different.

- If the difficulty is with an embedded URL, make sure that special symbols used in the text string have been encoded (as is described in APPENDIX B: CHARACTER SETS AND TRANSLATION). Especially check that all ampersands used before a field name are encoded (not necessary before a Lasso tag). If this isn't the case, you may run into problems when the "&" completes an HTML character reference. For example, if a field named "copy material" is used in an embedded URL string after the "&" symbol, it would appear on the HTML as

  ```
  ...&copy material=[field:editor]&...
  ```

  and be translated by the browser as the copyright symbol (©). Although it is missing the ending semi-colon (;) for the proper character reference (which uses the pattern &nnn;), some browsers (notably Netscape Navigator) will translate it anyway.

- If there are value lists, check to see if they are causing the problem by trying the action without the value list.
- As a last resort, try to reduce the number of fields to only those necessary for an -add or -search. Try creating a simplified version of the format file to add to or search one or just a few fields, and create a FileMaker database layout specifically for this test (don't forget to change the format file to reflect the change).
- If you have gone through all of these steps and still cannot get a Lasso response, it may be possible that the database is damaged. Refer to FILEMAKER PRO DATABASE IS DAMAGED for more information.

Lasso Does Not Add Information to a Specific Field

It appears that although the -add or -update action occurs, Lasso fails to add information to a specific field. If it is attempted to add to a field that is not on the layout specified by the layout command, no error will be displayed, but the data is lost.

Solution: Make sure that the FileMaker Pro layouts specified with the -layout command include all fields used in the specified format file. Confirm that there are no spaces either before or after the field name. In addition, check that data can be entered into the field on the specified FileMaker Pro layout. Open the "Format Fields" option for the various fields and make sure that "Allow entry into field" is selected. If this doesn't work, try to create a layout in FileMaker Pro that has all fields on it (when you go to "New Layout" choose "Single page form") and then specify that layout in the format file.

Unexpected Search Results

A search returns records when no values have been entered into fields, the expected behavior being a "No Records Found" error. Or, records are returned that should not have been.

Solution: If there are problems with a search, try to search directly within the FileMaker Pro interface to check the results. The FileMaker Pro database layout must contain all fields that are returned in the hitlist format file. Lasso will read all the fields in the specified layout with the layout tag on the search page; therefore, don't include any fields that are not used for the search para-

meters or hitlist page (these can be shown on a subsequent detail page by changing the layout referenced in the hitlist page for the [detail_link: …] tag).

One other reason for unexpected results could be due to having the last HTML "input" being set to a field instead of the submit action. This is a problem with certain browsers. To solve this, add a hidden action at the end of the form. The hidden action is formatted as follows:

```
<input type="hidden" name="-search">
```

This occurs because some browsers will incorrectly add a "return" to the last field in the form (if the last item is a text field), rather then an action request.

Another option is to always include the action in the HTML form that specifies Lasso. This is accomplished as follows:

```
<form action="action.lasso?-search" method="post">
```

For more details see CHAPTER 4: LASSO METHODOLOGY.

Another problem has been noted in which Internet Explorer 2.0 incorrectly inserts a space into the value of a form element which contains no value. This is a problem only when the "equals" operator is used on that field. In other words, in order for a selection list to not have any items selected for a search, a null value (value="") is used so that no value can be selected for that field. Internet Explorer 2.0 will send a space, so the search will seek records that have a space in that field. The solution is to use the "begins with" operator.

Searching Appears to be Case-Sensitive

If you are having trouble with searching in which the search parameters seem to be case-sensitive, check how the fields are stored in the FileMaker Pro index. If the fields are defined to be stored as ASCII, searching will be case-sensitive.

Solution: From within the "Define Fields" window, select "Options," then select the button in the lower left labeled "Storage options" and change the indexing language to anything other than ASCII. The searching should now be non–case-sensitive.

Pop-up List Does Not Work Properly

There are times when the FileMaker Pro layout gets "confused" about the values associated with a particular field, especially if a lot of changes to the layout have occurred, lists are redefined, or changes made to which field a value list is associated with.

Also, while field names are not case-sensitive, value list names are case-sensitive. If value list names do not match exactly (including case), then data will not be entered correctly. This does not apply on "post-Lasso" format files (using [value_list: …]…[option: …]…[/value_list] or [value_list: …]…[list_value]…[/value_list]) because their values are dynamically generated.

Solution: If the value list is hard-coded (not returned by Lasso from the database) and the value list is specified in FileMaker Pro, make sure that the names on the list match the items in the FileMaker Pro database EXACTLY. Better yet, if you're not returning the value list with Lasso then it isn't necessary to format the field as having a value list on the FileMaker Pro layout. Values can be entered into a standard FileMaker Pro field using any HTML selection list (such as a pop-up).

Try duplicating the layout being used, rename it, and specify the new layout in the add format file. Or, recreate any fields with selections lists on the layout in FileMaker Pro. Drag the field from the left side layout bar in FileMaker Pro and make sure it is associated with the correct value list. Do not create this field by copying another field or option-dragging another field and then changing the associated value list.

Only the First Option in a Pop-Up List is Displayed

There is an obscure oddity that may occur when the first field in the tab order of a layout has a value list. In other words, a layout in FileMaker Pro has the first item in the tab order as a field formatted to use a value list. When Lasso attempts to create a pop-up list using this field on a Lasso response page, it is unable to show more than the first value on the list.

Solution: Format the first field of the tab order as a "standard" field.

FileMaker Pro Database is Damaged

At times it is necessary to repair a FileMaker Pro database that was damaged as the result of the file being closed incorrectly, but is not damaged enough that FileMaker Pro gives the error alert that the database file must be recovered. It may also be necessary if a database was received as an email attachment. If changes were recently made and fields formatted with selection lists do not behave properly, or if the database is returning records incorrectly from a find or sorting in an unexpected order, then you may want to try to repair it. In addition, it may be a good routine maintenance practice. According to a Claris Tech Info document:

> "…most databases are good candidates for a routine compression. Saving a compressed copy rewrites the entire database, fitting as much data into each block as is possible. This procedure not only reclaims unused space in the file, it also rebuilds the file's structure."

Solution: If the FileMaker database is slightly corrupted it needs to be rebuilt. To repair a slightly damaged database and rebuild the database file follow these steps: save a clone of the database, recover the clone, open the recovered clone and save a compressed copy. Then repeat these steps one or two more times on the new compressed copy. Use the final, last, doubly cloned and recovered copy as the new database and import any existing records in from the original. This routine should force FileMaker Pro to truly rebuild the database structure.

Add Results in Failed Search

One of the most common problems with Lasso troubleshooting is when an -add or -update results in a failed search. Previous to Lasso 2.0, if any Lasso request was submitted without an action, Lasso would default to the -search action. If Lasso Security was not initialized and not present, a failed action would result in the alert "No records found" with the parameters of a search listed, as if a search action was attempted and failed. If Lasso Security was enabled, the error resulted in a security violation, which was confusing when the security appeared to be set up correctly. With Lasso 2.0 the default action was changed to -nothing so the error would appear to be a response file called up without any action performed. For example, the response page does not display field values.

Both of these problems can occur when a complete "search argument" detailing the Lasso action is truncated and is not completely delivered for proper processing. The search argument is any

request for a Lasso action, including -add, -search, -update, -delete, -scripts, and so on. The reason behind the lost action is not clear; however, the usual reason is that some element of the form, or the embedded URL, is not correct.

Solution: First make sure one can completely fill out a form or complete the action directly within the FileMaker database. Then, go through the elements of the form or embedded URL to check how the fields, database, and layout are specified (as mentioned under earlier troubleshooting topics).

If that does not work, try the following :

- Eliminate the layout tag in the Lasso format file (or embedded URL).
- Turn off "multi-user" network access (if on).
- Duplicate the layout being used, rename it, and specify the new layout in the add format file.
- These problems may also be linked to a flaw in the FileMaker Pro database. See FILEMAKER PRO DATABASE IS DAMAGED.
- Place the action at the start of the sequence of tags; for example, when dealing with an embedded URL, try moving the action as follows:

```
<a href="Lasso.acgi?-search&-database=Employees&-layout=Detail&
-response=Detail+Format.html&-op=cn&First+Name=Michael&-maxRecords=5&
-skiprecords=10&-sortfield=Last+Name"> Search </a>
```

Or, on a form, place the action within the "form" HTML tag, within the "action" attribute, as follows:

```
<form
action="../Lasso.acgi?-search&-database=Employees&-layout=summary"
method="post">
   …rest of the HTML form
</form>
```

Or, for the Plug-in:

```
   <form
action="action.lasso?-search&-database=Employees&-layout=summary">
   …rest of the HTML form
</form>
```

FM Link Does Not Show Tags

After launching FM Link and selecting the "Lasso tags" tab nothing appears.

Solution: Make sure that the FM Link application and "Lasso Tags" document are located in the same folder and that the "Lasso Tags" name has not been changed. The "Lasso tags" file should only be altered using extreme care. If changes were made, try reinstalling a fresh copy of the "Lasso Tags" file that is distributed in the same folder as FM Link. In order for FM Link to parse this document correctly, each command needs to be structured in the following sequence:

1. NAME
return (no text or spaces)
2. HTML CODE
return (no text or spaces)
3. DESCRIPTION/HELP
return (no text or spaces)
return (no text or spaces)

If a "#" is placed at the start of a line, that line is ignored. To start a new line without throwing off the sequence, put a single space at the start of the line (before any returns or text). For the latest copy visit the Blue World Web site (http://www.blueworld.com).

You may also try allocating more memory to the FM Link application.

FM Link Does Not Show FileMaker Pro Database

After launching FM Link, nothing appears when selecting the various tabs to show FileMaker Pro database information.

Solution: Make sure that the FileMaker Pro database is open before opening FM Link. Make sure to start at the "Databases" tab and double-click the name to show the layouts, and then the layouts to show the fields. If you need to update the list, do so at the database tab and click through the sequence again. If you close a database while FM Link is open the database will not be available, and FM Link may not work. Although the fields are still showing, it is necessary to open the database and update the list to have the fields show properly.

Form Submission Fails After Updating to New Version

After updating to a newer version of Lasso, actions such as -add, -update, or -search fail to successfully add or find data in the FileMaker Pro database. In versions of Lasso prior to 1.1, a different syntax was used to specify the database and layout. These early versions allowed for database and layout values to be specified on the reply/response page using the now obsolete [database: name-of-database] and [layout: name-of-layout] tags. This approach will no longer work with versions of Lasso greater than 1.1.

Solution: To correct, simply add the database and layout commands tags to the originating form or embedded URL. The database and layout are always specified as hidden inputs on the form that specifies the action. The syntax used would appear as:

```
<input type="hidden" name="-database" value="YourDBName">
<input type="hidden" name="-layout" value="YourLayoutName">
```

In addition, delete the now obsolete [database: name-of-database] and [layout: name-of-layout] tags from the response forms.

Portal Does Not Display Related Fields

When you are having trouble returning related information contained within a portal check the following:

- The relationship name and all related fields should not have any trailing space or special character in their name (not the field data).
- All related fields within the portal should be completely within the first row that shows as a blank white space in FileMaker Pro's layout view.
- If you specify that a related field is within a portal, do not also include it outside the portal on the FileMaker Pro layout. And vise versa.
- Try to enter values into all fields within the portal to make sure that data can be entered. Also check that none are auto-entered by the related database, or are calculation fields.

Appendix D: Error Codes

If Lasso does not appear to be working as one might expect, check the log window to see if any errors have been reported. The log is visible when Lasso is running. When reporting any errors for Lasso support, copy these error messages from the log window (the error code number plus its error text constitute the full error reference). Note that the Lasso Plug-in will display error messages in the log window of whichever Web server is being used.

Error code messages are returned for known errors. Unknown errors are reported as "An unexpected error occurred." The following are some of the known errors you may encounter along with possible solutions.

Error Code: -1 Couldn't Find Format File

Description: Lasso is unable to perform an action. This will occur if an earlier version of Lasso is on the server, but more recent Lasso tags are used, in particular the -response tag. -response is synonymous with either [filename] or [format], and all will work with version 1.1 of Lasso. [filename] or [format] were used with Lasso Lite and/or earlier versions of Lasso. -response is the preferred command tag and future versions of Lasso may not support [filename] or [format].

Solution: Update Lasso on the server.

Error Code: -1 Out of memory!

Description: Lasso has used up its normal memory allocation and is running on reserve memory.

Solution: Quit and restart Lasso. Contact Blue World technical support and report the error. Try to allocate more memory to the Lasso.acgi, or to the Web server if using the Lasso Plug-in.

Error Code: -2 Out of memory! Click anywhere to quit...

Description: This is a fatal error caused by the complete exhaustion of Lasso's reserve memory.

Solution: Click the mouse button to cause Lasso to quit. Restart Lasso. Try to allocate more memory to the Lasso.acgi, or to the Web server if using the Lasso Plug-in. Contact Blue World technical support and report the error.

Error Code: -35 Could not find format file

Description: The format file specified by a -response or -deletereply command could not be found. This error usually occurs if the format file is located on a shared volume but is not available.

Solution: Make sure that the volume is mounted. For example, set up AppleShare to mount the volume automatically when the computer is restarted.

Error Code: -39 Could not find format file

Description: Lasso fails to return a format file after an action. A format file is specified by a -response or deletereply command. Lasso will try three times before reporting error -43 when opening format files.

Solution: Make sure the path to the file is specified properly. Check that the exact spelling of the file name has been used. Check that there are no stray spaces after the name, try to copy and paste the name directly from the file into your HTML form. Also, in certain instances a very full and fragmented drive may make it difficult for the format file to be found on the drive. Clear files on the drive if there is less than 10MB of space left on the drive. Also try to defragment the drive using a drive utility (such as Norton Speed Disk©.

Error Code: -43 Could not find format file

Description: This is very similar to a -35 error; refer to that error for more information.

Error Code: -49

File is open

Description: Format file is open and cannot be referenced by Lasso.

Solution: Make sure the file is not open by another application.

Error Code: -50

No valid action in input form

Description: Lasso did not find an action command in the input data received with the request. Action tags are: -show, -search, -add, -update, -delete, -scripts, -random, and -duplicate. (Other action tags may be defined in the future).

Solution: Make sure one of the action tags appears in your HTML form or in the URL if you"re hard-coding a Lasso request. This error can also appear when using the CGI and more than 24K bytes of data is sent with a request. This can happen if the user enters a large amount of data in a text area field in an HTML form.

Error Code: -50

Format File not found

Description: The return format file is not displayed. Usually this occurs when a hitlist link is selected, that is, when specific record details are requested from a hitlist. It also occurs when a link is used that has embedded Lasso commands. The [detail_link: …] tag is used as follows:

```
<a href="[detail_link: layout=detail, format=
"path_to_file/Detail%20Format.html"]"> [field: "Field Name"]</a>
```

Or:

```
<a href="Lasso.acgi?-database=database name&-layout=layout name&
-response=name.html&-operator=eq&Insert Field Name=value_in_Field&
-sortfield=Insert Field Name&-search"> Search </a>
```

Solution: Spaces that are used in file names must be encoded with either a plus sign or by "%20." For more information see APPENDIX B: CHARACTER SETS AND TRANSLATION. It is also good practice to name all of your files, databases, layouts, and field names without spaces or special symbols.

Check whether the embedded URL or [detail_link: …] tag contains any encoded spaces. The encoded character for a space (%20) will

prevent any URL from operating if placed incorrectly. For example, the following embedded URL will not work correctly until the spaces are encoded as follows:

```
<a href="Lasso.acgi?-database=database%20name&-layout=layout%20name&
-response=name.html&-op=eq&Insert%20Field%20Name=value_in_Field&
-sortfield=Insert%20Field%20Name&-search"> Search </a>
```

On the other hand, an embedded link will not work correctly if an HTML authoring program has encoded all of the spaces and changed this to:

```
<a href="[detail_link:%20layout=detail,%20format=path_to_file/Detail%20Format.
html]">
[field: "Field Name"]</a>
```

Error Code: -50 Object not found

Description: There is a peculiarity when the search form contains only one text input field. In this case, the user can hit the "Enter" key rather than click on the submit button. In this case, the -search parameter associated with the submit button doesn't get sent to Lasso, since the submit button serves to send the action. In other words, if the action is specified through the submit button then the action is not sent unless the button is selected.

Solution: The solution in this case is to add a hidden action at the end of the form:

```
<input type="hidden" name="-search">
```

The value can be anything, or blank. This will insure that the -search parameter is sent if the user presses the "Enter" key. If the user clicks the submit button the -search parameter will get sent twice, but that won't hurt anything.

Another option is to always include the action in the HTML form that specifies Lasso. This is accomplished as follows:

```
<form action="action.lasso?-search" method="post">
```

For more details see CHAPTER 4: LASSO METHODOLOGY.

Again, this is only needed if the search form contains only a single text input field. The form can contain any number of pop-up (other selections lists) or text area fields. If there are more text inputs, pressing the Enter key won't cause the form to be submitted.

Error Code: -108 Not enough memory for request

Description: Lasso did not have enough memory available to finish processing the request.

Solution: This error is most likely to occur when a very large amount of data is returned from FileMaker Pro as the result of a search request. Check to make sure the layout specified in your search request contains only the fields needed to conduct the search and format the results (Lasso retrieves data for ALL fields in a layout, even if some fields aren't used in the format file). Try reducing the maximum number of records the user is allowed to return as the result of a search. Finally, increase Lasso's memory partition in the Get Info box.

Error Code: -120 Could not find format file

Description: The directory (folder) specified in the path in a -response or -deletereply command could not be found.

Solution: Check the spelling of all directories (folders) in the path. Make sure the path is specified properly.

Error Code: -192 Could not get SIZE -1 resource

Description: The Lasso.acgi application file is corrupted.

Solution: Reinstall Lasso.acgi from the backup, or download and reinstall a new copy from Blue World's Web site.

Error Code: -700 Could not find email format file

Description: The format file specified by an -emailformat command could not be found.

Solution: Check the spelling of the file name. Make sure the path to the file is specified properly.

Error Code: -701 All email tags must be assigned a value

Description: In order for an email notification to be sent, all five of the email parameters (-emailhost, -emailfrom, -emailto, -emailsubject, and -emailformat) must be specified.

Solution: Make sure you've specified values for all five parameters in your HTML form. Make sure the parameter names are spelled correctly.

Error Code: -1701 Unexpected error code

Description: Incorrect syntax for a routine; in particular, the repeating fields routine.

Solution: See CHAPTER 10: REPEATING FIELDS AND RELATED FIELDS for details on what might trigger this error.

Error Code: -1701 Could not get Error String!

Description: FileMaker Pro may be running out of memory, or an incompatible version of the Object Support Library System extension may be in use.

Solution: Quit FileMaker Pro and find the application in the Finder. Use the "Get Info" command to assign more memory to the application. Also, check the version of "ObjSupportLib" and refer to CHAPTER 2: INTRODUCTION.

Error Code: -1712 Apple Event timed out

Description: Usually this is seen when a FileMaker Pro script fails to execute. In this case the Apple Event that interacts with the Finder is unable to make FileMaker Pro the frontmost application. FileMaker Pro must come to the front in order for the script to run.

Solution: Check that the "Finder Scripting Extensions" is installed and loaded when the system is started. It is required for the -scripts tag to work. Also check to see if some other extension is preventing FileMaker from activating and becoming the frontmost application while the script is being executed.

Error Code: -5550 or -5551 Unexpected Gestalt error

Description: The Lasso.acgi application file is corrupted.

Solution: Reinstall Lasso.acgi from your backup, or download and reinstall a new copy from Blue World's Web site.

Error Code: -10004 Security violation

Description: A security error occurs after an "add" or "search" form is submitted. This error pertains to passwords that need to be set in the FileMaker Pro database and not those set through Lasso Security.

Solution: Make sure the database served by Lasso has been opened with a password that allows changes to be made to the database. At a minimum, a password is needed that permits "Browse" "Print," and "Export" for searching the database. To allow records to be added, permission to "Create Records" is also needed. In addition, check whether the password is assigned to a "Group" that controls access to fields or layouts. The password used to open the database for Lasso needs to have at least read-only access to all fields and layouts in the database.

Error Code: -10005 Read Access Denied

Description: This error is with passwords set in the FileMaker Pro database and not those set by Lasso Security. This error occurs whenever trying to access a database that has been opened with a password that does not have read access permission for all fields and layouts.

Solution: Open the FileMaker database and select "Access Privileges" in the Edit menu, then "Define Groups." If no groups have been established, select the password used to open the database for use with Lasso. Check that permissions have been assigned for at least read-only access to all fields and layouts in the database.

Error Code: -10006 Write access denied

Solution: Make sure the fields referenced in the Lasso format file match FileMaker Pro field names EXACTLY. Case sensitivity DOES NOT apply unless specifying value lists.

Appendix E: Support

Technical Support

- Registered users receive free telephone and email support during normal business hours for 45 days from date of original purchase. Free support does not apply to upgrades.
- Extended support and program upgrades are subject to charge. Extended support is billed at $25 per quarter hour via authorized credit card.

To help facilitate accurate and timely technical assistance, the following information is required:

- Type and configuration of computer (e.g., Performa 5215, 16 MB RAM, 1.2 GB hard drive)
- Version of Mac OS
- Version and edition of Lasso
- Version of FileMaker Pro
- Excerpts from error log window indicating errors
- List of other software running on the system if even remotely applicable to problems you're experiencing (include version numbers).

Contact Information

Lasso Web Site: http://www.blueworld.com/Lasso/

Email Address: support@blueworld.com

FAX: 425-646-0236

Phone: 425-646-0288 (9 am–5 pm PST)

Address:

Blue World Communications, Inc.
10900 NE 8th Street, Suite 1525
Bellevue, WA 98004 U.S.A.

Lasso Talk

To stay informed about the evolving capabilities of Lasso, please join the Lasso Talk email discussion group. This forum is frequented by hundreds of Lasso users who exchange tips and techniques on how to better use Lasso. Email discussion forums can often be the best way to receive timely support and tips for better use of your software. Likewise, searching for answers in Web-based archives for an email discussion list often yields quick answers. Note: Posting a message to Lasso Talk does not ensure a response from Blue World.

To subscribe, send an email message to Lasso@blueworld.com with SUBSCRIBE in the subject line (text in the body is ignored).

The searchable archives for Lasso Talk are available at:

http://www2.blueworld.com/lists/Lasso/search.html

Documentation Updates

At Blue World Communications we value the importance of quality and up-to-date documentation. Announcements on new versions of the documentation will be made on the Lasso Talk email list.

If you find any errors or have suggestions as to how we may better improve our documentation, please contact us at:

documentation@blueworld.com.

To ensure you have the most current version of the documentation, please periodically check the online version of the Lasso documentation at:

http://www.blueworld.com/Lasso/User_Guide/

There may be periodic minor updates to the documentation which are not announced.

The Lasso Web site, including all of the online documentation, is also searchable via the following URL:

http://www.blueworld.com/lasso/lsearch/search.lasso

Appendix F: Usage Rights

Please Do Not Make Illegal Copies of This Software

The software you are using was produced through the blood, sweat, and tears of many people: designers, artists, programmers, distributors, retailers, and other dedicated workers. The costs of developing this and other software programs are recovered through software sales. The unauthorized duplication of software raises the cost for all legitimate users.

This software is protected by federal copyright law. Your cooperation in upholding the law will insure continued low-cost upgrades and new software. Copying software for any reason other than to make a backup is a violation of federal laws. Individuals who make illegal copies of software may be subject of civil and criminal penalties.

Software Publishers Association
11011 Connecticut Ave., NW, Suite 901
Washington, DC 20036

Lasso License Agreement

PLEASE READ THIS LICENSE CAREFULLY BEFORE INSTALLING OR USING Lasso© SOFTWARE. BY DOWNLOADING, INSTALLING, AND/OR USING THE SOFTWARE, YOU ARE AGREEING TO BE BOUND BY THE TERMS OF THIS LICENSE. IF YOU DO NOT AGREE TO THE TERMS OF THIS LICENSE, YOU ARE NOT AUTHORIZED TO DOWNLOAD, INSTALL, AND/OR USE THIS SOFTWARE.

1. License. The Lasso application software accompanying this License, whether on disk, in read only memory, or on any other media or networked storage device (the "Lasso Software") and related documentation (the "data") are licensed to you by Blue World Communications (the "Author").You own the disk on which the Lasso Software and data are recorded but the Author and/or the Author's Licenser(s) retain title to the Lasso Software and related documentation.

   This License allows you to install and use a single copy of the Lasso Software and data on a single computer and make one copy of the Lasso Software and data in machine-readable form for backup purposes only. You must reproduce on such copy the Author's copyright notice and any other proprietary legends that were on the original copy of the Lasso

Software and related documentation. You may transfer all your license rights in the Lasso Software and related documentation, the backup copy of the Lasso Software and related documentation, and copy of this License to another party, provided the other party reads and agrees to accept the terms and conditions of this License and a transfer of ownership letter, listing contact information for both parties, is sent to the Author.

2. Restrictions. The Lasso Software and related documentation contains copyrighted material, trade secrets and other proprietary material and in order to protect them you may not decompose, reverse engineer, disassemble or otherwise reduce the Lasso Software to a human-perceivable form without the Author's written permission.

3. Termination. This License is effective until terminated. You may terminate this License at any time by destroying the Lasso Software and related documentation and all copies thereof. This License will terminate immediately without notice from the Author if you fail to comply with any provision of this License. Upon termination you must destroy the Lasso Software, and related documentation and all copies thereof.

4. Disclaimer of Warranty on Lasso Software. You expressly acknowledge and agree that use of the Lasso Software and related documentation is at your sole risk. The Lasso Software and related documentation are provided AS IS and without warranty of any kind and the Author EXPRESSLY DISCLAIMS ALL WARRANTIES, EXPRESS OR IMPLIED, INCLUDING, BUT NOT LIMITED TO, THE IMPLIED WARRANTIES OF MERCHANTABILITY AND FITNESS FOR A PARTICULAR PURPOSE. THE AUTHOR DOES NOT WARRANT THAT THE FUNCTIONS CONTAINED IN THE LASSO SOFTWARE WILL MEET YOUR REQUIREMENTS, OR THAT THE OPERATION OF THE LASSO SOFTWARE WILL BE UNINTERRUPTED OR ERROR-FREE, OR THAT DEFECTS IN THE LASSO SOFTWARE WILL BE CORRECTED.

   FURTHERMORE, THE AUTHOR DOES NOT WARRANT OR MAKE ANY REPRESENTATIONS REGARDING THE USE OR THE RESULTS OF THE USE OF THE LASSO SOFTWARE AND DATA OR RELATED DOCUMENTATION IN TERMS OF THEIR CORRECTNESS, ACCURACY, RELIABILITY, OR OTHERWISE. NO ORAL OR WRITTEN INFORMATION OR ADVICE GIVEN BY THE AUTHOR OR A REPRESENTATIVE AUTHORIZED BY THE AUTHOR SHALL CREATE A WARRANTY OR IN ANY WAY INCREASE THE SCOPE OF THIS WARRANTY. SHOULD THE LASSO SOFTWARE PROVE DEFECTIVE, YOU (AND NOT THE AUTHOR OR A REPRESENTATIVE AUTHORIZED BY THE AUTHOR) ASSUME THE ENTIRE COST OF ALL NECESSARY SERVICING, REPAIR OR CORRECTION. SOME JURISDICTIONS DO NOT ALLOW THE EXCLUSION OF IMPLIED WARRANTIES, SO THE ABOVE EXCLUSION MAY NOT APPLY TO YOU.

5. Limitation of Liability. UNDER NO CIRCUMSTANCES INCLUDING NEGLIGENCE, SHALL THE AUTHOR BE LIABLE FOR ANY INCIDENTAL, SPECIAL OR CONSEQUENTIAL DAMAGES THAT RESULT FROM THE USE OR INABILITY TO USE THE LASSO SOFTWARE OR RELATED DOCUMENTATION AND DATA, EVEN IF THE AUTHOR OR AN AUTHORIZED REPRESENTATIVE OF THE AUTHOR HAS BEEN ADVISED OF THE POSSIBILITY OF SUCH DAMAGES. SOME JURISDICTIONS DO NOT

ALLOW THE LIMITATION OR EXCLUSION OF LIABILITY FOR INCIDENTAL OR CONSEQUENTIAL DAMAGES SO THE ABOVE LIMITATION OR EXCLUSION MAY NOT APPLY TO YOU. In no event shall The Author's total liability to you for all damages, losses, and causes of action (whether in contract, tort (including negligence) or otherwise) exceed the amount paid by you for the Lasso© Software and data.

Complete Agreement. This License constitutes the entire agreement between the parties with respect to the use of the Lasso Software, related documentation and data, and supersedes all prior or contemporaneous understandings or agreements, written or oral, regarding such subject matter. No amendment to or modification of this License will be binding unless in writing and signed by the Author or a duly authorized representative of the Author.

clip2gif Disclaimer

GIF image encoding Copyright© 1989 by Jef Poskanzer.

Permission to use, copy, modify, and distribute this software and its documentation for any purpose and without fee is hereby granted, provided that the above copyright notice appear in all copies and that both that copyright notice and this permission notice appear in supporting documentation. This software is provided "as is" without express or implied warranty.

The Graphics Interchange Format© is the Copyright property of CompuServe Incorporated. GIF(sm) is a Service Mark property of CompuServe Incorporated.

clip2gif is a freeware product. As such it is not supported by its author (see legal terms in the clip2gif distribution). Because we do not have access to the source code, Blue World Communications, Inc. cannot support use of clip2gif with Lasso. BLUE WORLD COMMUNICATIONS, INC. DOES NOT PROVIDE ANY WARRANTY FOR USE OF CLIP2GIF WITH LASSO. YOU ASSUME ALL LIABILITY FOR RISK OF USE OF CLIP2GIF WITH LASSO. If you have any doubts about this, do not use clip2gif to serve FileMaker Pro pictures with Lasso.

Appendix G: LDML 2.5 Quick Reference Chart

Tag	Tag Name	Tag Type
-add	Add Record	Action
-adderror	Add Error Reply	Command
[begnum]	Beginning Number	Substitution
[checked]	Checked	Sub-Container
[client_addr]	Client Domain Address	Substitution
[client_ip]	Client IP Address	Substitution
[client_password]	Client Password	Substitution
[client_type]	Client Browser Type	Substitution
[client_username]	Client Username	Substitution
[content_type: …]	Content Type	Substitution
[cookie: …]	Display Cookie	Substitution
-database	Specify Database	Command
[database_name]	Display Database Name	Substitution
[db_name]	Display Open Database	Sub-Container
[db_names]	Display All Open Databases	Container
[decode_url: …]	Decode URL	Substitution
-delete	Delete Record	Action
-deletereply	Delete Reply	Command
[detail_link: …]	Detail Link	Substitution
-doscript.post	FMP Script Post-Lasso	Command
-doscript.post.back	FMP Script Post-Lasso in Background	Command
-doscript.pre	FMP Script Pre-Lasso	Command
-doscript.pre.back	FMP Script Pre-Lasso in Background	Command
-doscript.presort	FMP Script Pre-Sort	Command
-doscript.presort.back	FMP Script Pre-Sort in Background	Command
-duplicate	Duplicate Record	Action
-duplicatereply	Duplicate Reply	Command
[else]	Else	Sub-Container
[else: if: …]	Else If	Sub-Container
-emailbcc	Email BCC	Command
-emailcc	Email CC	Command
-emailformat	Email Format	Command
-emailfrom	Email From	Command
-emailhost	Email Host	Command
-emailsubject	Email Subject	Command
-emailto	Email To	Command
[encode_breaks: …]	Encode Breaks	Substitution
[encode_html: …]	Encode HTML	Substitution
[encode_raw: …]	Encode Raw	Substitution
[encode_smart: …]	Encode Smart	Substitution
[encode_url: …]	Encode URL	Substitution
[endnum]	Ending Number	Substitution

Tag	Tag Name	Tag Type
[event: …]	Apple Event	Container
[event_errorstring]	Event Error String	Sub-Container
[event_result]	Event Result	Sub-Container
[event_resultcode]	Event Result Code	Sub-Container
[field: …]	Field	Substitution
[field_name: …]	Display Field Name	Substitution
-findall	Find All Records	Action
[form_param: …]	Form Parameter Value	Substitution
[header]	HTTP Header	Container
[html_comment]	HTML Comment	Container
[if: …]	Conditional Statement	Container
-image	Image Action	Action
[image: …]	Image	Substitution
[include: …]	Include a File	Substitution
[inline: …]	Inline	Container
[inline_result]	Inline Result	Substitution
[lasso_action]	Display Lasso Action	Substitution
[lasso_process: …]	Lasso Process	Substitution
-lassopassword	Submit Password	Command
-lassousername	Submit Username	Command
[lay_name]	Display Database Layout	Sub-Container
[lay_names: …]	Display All Layout Names	Container
-layout	Specify Layout	Command
[layout_name]	Display Layout Name	Substitution
[list_value]	Display List Values	Sub-Container
[log: …]	Log Activity	Container
-logicalop	Logical Operator	Command
[logicalop_value]	Display Logical Operator	Substitution
[loop: …]	Loop	Container
[loop_count]	Loop Count	Sub-Container
[math-add: …]	Math-Add	Substitution
[math-div: …]	Math-Divide	Substitution
[math-mod: …]	Math-Modulo	Substitution
[math-mult: …]	Math-Multiply	Substitution
[math-round: …]	Math-Round Result	Substitution
[math-sub: …]	Math-Subtract	Substitution
-maxrecords	Maximum Records	Command
[maxrecords_value]	Display Maximum Records Value	Substitution
[next]	Next Record Group	Container
[next_url]	Display Next URL	Substitution
[nfound]	Number Records Found	Substitution
-noresults	No Results Reply	Command
-nothing	Nothing	Action
[nshown]	Number Record Shown	Substitution
-opbegin	Begin Logical Operator	Command
-opend	End Logical Operator	Command
-operator	Operator	Command
[option: …]	Options in Selection List	Substitution

Tag	Tag Name	Tag Type
[portal: …]	Portal	Container
[post_inline: …]	Post-Inline	Substitution
[prev]	Previous Record Group	Container
[prev_url]	Display Previous URL	Substitution
-random	Find Random Record	Action
-recid	Record ID	Command
[recid_value]	Record ID Value	Substitution
[record]	Record Display	Container
[referrer]	Referrer	Container
[referrer_url]	Display Referrer URL	Substitution
[repeat_value]	Display Repeat Value	Sub-Container
[repeating: …]	Repeating Fields	Container
[repetition]	Repetition	Sub-Container
-reqfieldmissing	Required Field Missing	Command
-required	Required Field	Command
-response	Response	Command
[response_file_path]	Display Response File Name	Substitution
-scripts	Execute FileMaker Pro Script	Action
-search	Search	Action
[search_args]	Display Search Arguments	Container
[search_field]	Display Field Searched	Sub-Container
[search_op]	Display Search Operator	Sub-Container
[search_value]	Display Search Value	Sub-Container
[selected]	Selected	Sub-Container
[server_date]	Display Current Date	Substitution
[server_day]	Display Current Day	Substitution
[server_time]	Display Server Time	Substitution
[set_cookie: …]	Set a Cookie Value	Substitution
[set_var: …]	Set a Variable	Substitution
-show	Show Record	Action
-skiprecords	Skip Returned Records	Command
[skiprecords_value]	Display Skip Records Value	Substitution
[sort_args]	Display Sort Parameters	Container
[sort_field]	Display Sort Field	Sub-Container
[sort_order]	Display Sort Order	Sub-Container
-sortfield	Sort Field	Command
-sortorder	Sort Order	Command
[string_concatenate: …]	String Concatenate	Substitution
[string_countfields: …]	String Count Fields	Substitution
[string_extract: …]	String Extract	Substitution
[string_findposition: …]	String Find Position	Substitution
[string_getfield: …]	String Get Field	Substitution
[string_insert: …]	String Insert	Substitution
[string_length: …]	String Length	Substitution
[string_lowercase: …]	String Lower Case	Substitution
[string_remove: …]	String Remove	Substitution
[string_removeleading: …]	String Remove Leading	Substitution
[string_removetrailing: …]	String Remove Trailing	Substitution
[string_replace: …]	String Replace	Substitution
[string_uppercase: …]	String Upper Case	Substitution

Tag	Tag Name	Tag Type
-timeout	Timeout	Command
-token	Set Token	Command
[token_value]	Display Token Value	Substitution
[total_records]	Display Total Records	Substitution
-update	Update Record	Action
[value_list: …]	Value Lists	Container
[var: …]	Retrieve a Variable	Substitution
[while: …]	While	Container

Appendix H: Index

D

E

F

G

H

I

J

O

P

Q

R

S

T

U

V

W